算法与数据结构（C语言）

**参考资料：**

《国嵌出品\_数据结构深度实战专题班》

算法分析与设计 的课程（查找资料）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本记录 | | | | |
| 版本号 | 日期 | 修改章节 | 修改内容及说明 | 编制者 |
| V1.0 | 2016/10/19 |  |  | 查锦 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 简介

## 1 宗旨

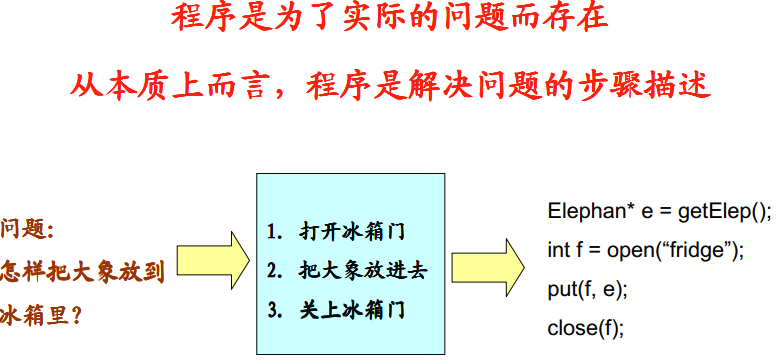
本文档主要用于记录在学习算法与数据结构过程中的思想，重点在于设计。好的设计才是程序实现的灵魂。

## 2 编译器

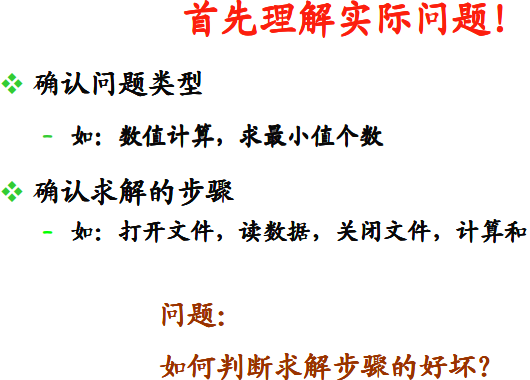
Dev-Cpp 5.2.0.3 MinGW32

# 一 进阶高手的大门

## 1 理解程序的本质



## 2 一小步的进阶



## 3 程序评鉴初探



审判——什么是好的代码？不能简单的认为，应当有一个判断的标准。

## 4 小结

1. 程序是为了具体的问题而存在的
2. 程序需要围绕问题的解决进行设计的
3. 同一个问题有多重解决方案

**如何追求程序的“性价比”？是否有可量化的方法判断程序的好坏？**

# 二 数据的艺术

## 1 数据结构起源

1. 计算机从解决数值计算问题到解决生活中的问题
2. 现实生活中的问题涉及不同个体间的复杂联系
3. 需要在计算机程序中描述生活中个体间的联系

数据结构主要研究非数值计算程序问题中的操作对象以及它们之间的关系。

## 2 关键概念

**数据**：程序的操作对象，用于描述客观的事物

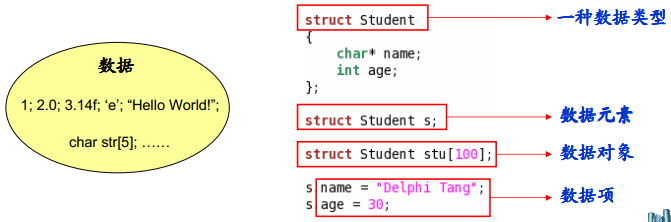
**数据的特点**

1. 可以输入到计算机
2. 可以被计算机程序处理。

**数据元素**：组成数据的基本单位

**数据项：**一个数据元素由若干数据项组成

**数据对象：**性质相同的数据元素的集合



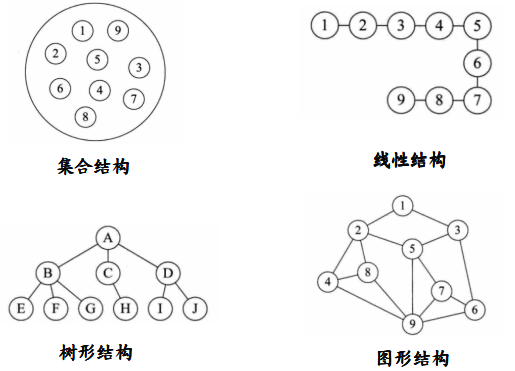
* **数据对象 -> 数据元素 -> 数据项**
* 数据元素之间不是独立的，存在特定的关系，这些关系即**结构**
* **数据结构指数据对象中数据元素之间的关系，**如：数组中各个元素之间存在固定的线性关系
* 编写一个“好”的程序之前，必须分析待处理问题中各个对象的特性，以及对象之间的关系。

数据之间有哪些关系（或者说结构）呢？前人总结如下：数据的逻辑结构和物理结构。

## 逻辑结构

**数据元素的关系如下：**

1. **集合结构** 数据元素之间没有特别的关系（没有任何关系），仅同属相同集合。
2. **线性结构** 数据元素之间是一对一的关系（类似数组）
3. **树形结构** 数据元素之间存在一对多的层次关系
4. **图形结构** 数据元素之间是多对多的关系



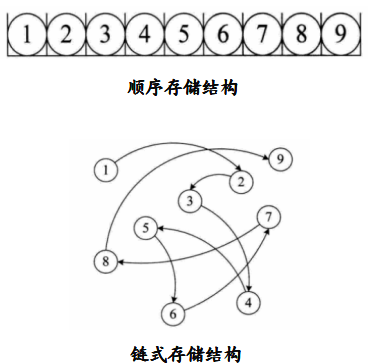
* 有了这些逻辑结构（数据结构的理论知识），这些图、树怎么让它在计算机中表现出来，怎么在计算机中存储这些结构呢？如果不能把这些关系输入计算机，数据结构的理论将一文不值，因此，了解完逻辑结构后，要掌握物理结构。

## 4 物理结构

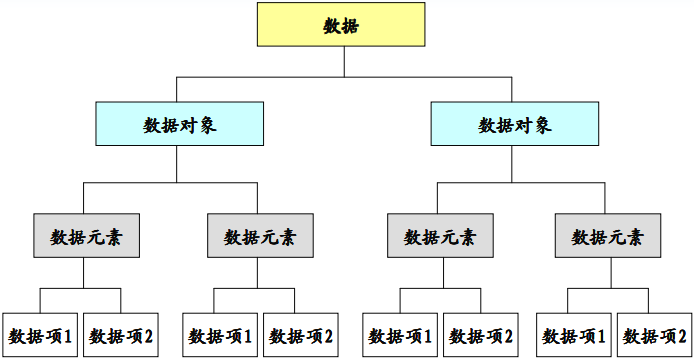
物理结构就是**逻辑结构在计算机中的存储形式**

1. **顺序存储结构** 将数据存储在地址连续的存储单元里（如数组）。
2. **链式存储结构** 将数据存储在任意的存储单元里，通过保存地址的方式找到相关联的数

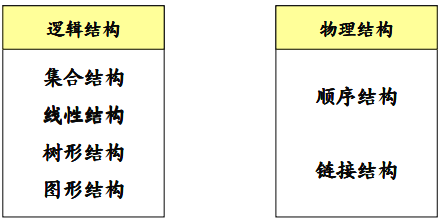
据元素（如链表的实现，逻辑相邻，物理不连续）。



## 5 小结



* **数据结构**是相互之间存在一种或多种特定关系的数据元素的集合。
* 按照视点的不同，**数据结构可以分为逻辑结构和物理结构。**



因此，数据结构这门课程中所有要研究的理论上的东西都是在研究逻辑结构。那为什么要说物理结构呢，因为实现这些逻辑结构的基础就是物理结构（即顺序结构和链接结构）。

物理结构是基础，逻辑结构是研究的重点。

# 三 程序的灵魂—算法

## 1 数据结构与算法

1. 数据结构只是静态的描述了数据元素之间的关系
2. 高效的程序需要在数据结构的基础上设计和选择算法

* **高效的程序：恰当的数据结构 + 合适的算法。**
* *算法分析与设计的课程，重点讲解算法，并从数学的高度来讲解一些算法，讲解算法对一些设计的经典模式，重点讲解算法引进数据结构。*
* *这门课程，重点讲解数据结构引进算法。*

## 2 算法的定义

1. 算法是特定问题求解步骤的描述
2. 在计算机中表现为指令的有限序列

**算法是独立存在的一种解决问题的方法和思想。**

*Explain：即便没有我们的程序语言，即便没有我们的计算机，我们的算法仍然可以存在。*

**对于算法而言，语言并不重要，重要的是思想。**

*Explain：有的人说，语言也不用深入的学，因为编程的时候，关键的是思想，而不是语言，但是，这句话的正确性是有前提条件的：对于我们设计一个算法而言语言并不重要，重要的是思想，但是对于软件开发而言，语言的选择是非常重要的，开发者对语言的掌握程度也是非常重要的。 例如：编写一个linux的驱动程序，当然用C语言，用一个JavaScript来编写是不可能的，那么这时候语言的选择是很重要的。 因此这句话是有前提条件的。*



## 3 算法的特性

算法有以下5条特性，缺一不可，缺少了一样即不叫算法，不成立。

1. **输入** 算法具有0个或多个输入
2. **输出** 算法至少有1个或多个输出

*例如：printf打印、输出文件、蜂鸣器等，不是狭义的屏幕打印东西。*

1. **有穷性** 算法在有限的步骤之后会自动结束而不会无限循环

*例如：QQ用了很多算法，但是其QQ本身并不是算法，是一个程序。*

1. **确定性** 算法中的每一步都有确定的含义，不会出现二义性（固定输入会固定输出）
2. **可行性** 算法的每一步都是可行的

有了上面5个特性，就一定是一个好算法吗？怎么判断一个算法的好坏，一个程序的好坏？那么算法是有设计的准则的，如下。

## 4 算法设计的准则

1. **正确性**

* 算法对于合法数据能够得到满足要求的结果
* 算法能够处理**非法输入**，并得到合理的结果
* 算法对于**边界数据和压力数据**都能得到满足要求的结果

注意： 正确性是算法最需要满足的基本的准则，但是作为计算机程序，不可能无限制的满足这条准则。

*Explain：对于最后一点，最后一点，算法几乎不可能达到。比如用经典的冒泡排序法，4千亿个整形数据来进行冒泡排序，我们的硬盘容量根本无法满足。那我们的冒泡排序不正确吗？根本不是，只是这个数据太特殊，我们一般也不需要处理这么大的数据，因此第三点不强制要求满足，能满足多少尽量满足多少，但是前两点必须满足，不然算法不成立。*

1. **可读性**

算法要方便阅读，理解和交流。

1. **健壮性**

算法不应该产生莫名其妙的结果。

*输入得到的结果有时候对，有时候错，那么说明我们的算法的健壮性需要加强。*

1. **高性价比**

利用最少的时间和资源得到满足要求的结果。

*这是我们要追求的。*

注意： 算法可读性是最容易被忽视的，然而，程序是写给人看的，而不是计算机。

*并不是看起来越复杂的就是越高级的。*

## 5 小结

* 算法是为了解决实际问题而设计的
* 数据结构是算法需要处理的问题载体
* 数据结构与算法相辅相成，缺一不可

**程序 = 数据结构 + 算法**

# 四 审判程序的灵魂

## 算法效率的度量

### 1）事后统计法

比较不同算法对同一组输入数据的运行处理时间。

**缺陷**

为了获得不同算法的运行时间必须编写相应程序；

运行时间严重依赖硬件以及运行时的环境因素；

算法的测试数据的选取相当困难；

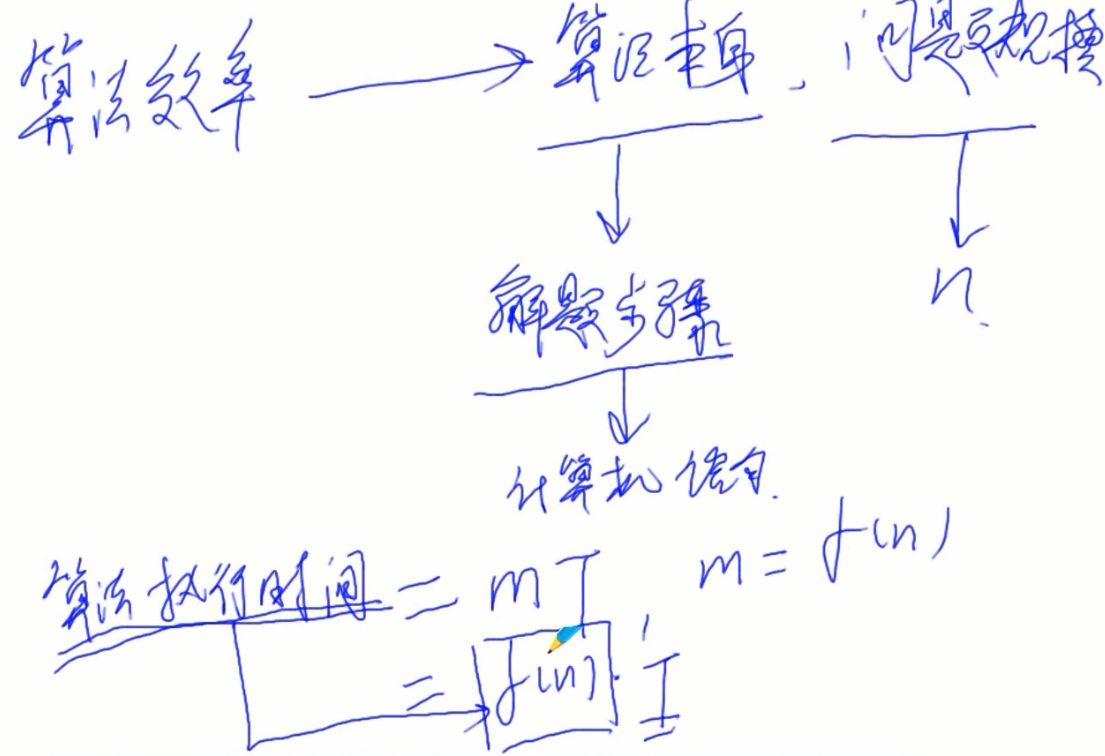
事后统计法虽然直观，但是实施困难且缺陷多，一般不予考虑。

### 2）事前分析估算

依据统计的方法对算法效率进行估算（采用）。

### 3）影响算法效率的主要因素

* 算法采用的策略和方法（算法本身，即解题步骤->反映到计算机就是计算机语句）
* 问题的输入规模
* 编译器所产生的代码
* 计算机执行速度



算法效率主要跟我们的算法本身和问题规模决定。算法本身也就是解题步骤（m），解题步骤的最后就是计算机语句，设问题规模为n。

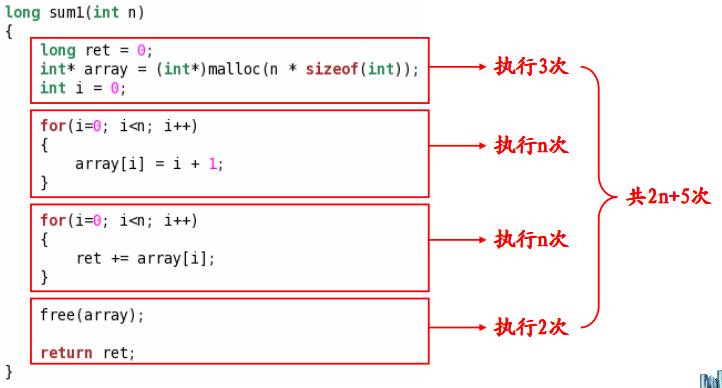
那么，算法的执行时间 = mT， 假定计算机执行每条语句的时间为固定的，其实解题步骤是问题规模的函数，m = f(n)，问题规模越大，解题步骤越多。

因此，算法的执行时间 = f(n).T，T一定。算法的问题演变为算法从开始到结束一共经历了多少步，每一步的时间是固定的，因此能估算出算法的执行时间。

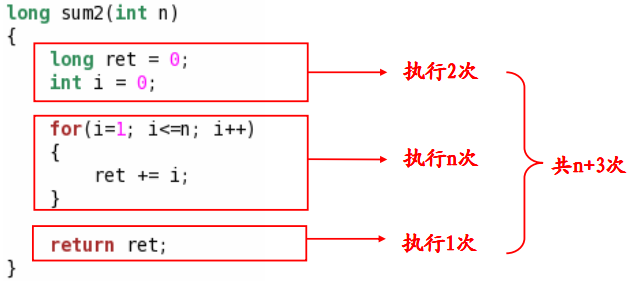
### 4）算法效率的简单估算

以求和算法为例：

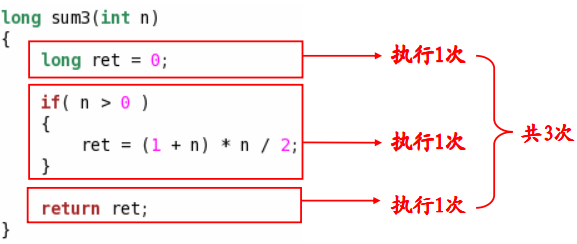
方法1：



方法2：



方法3：



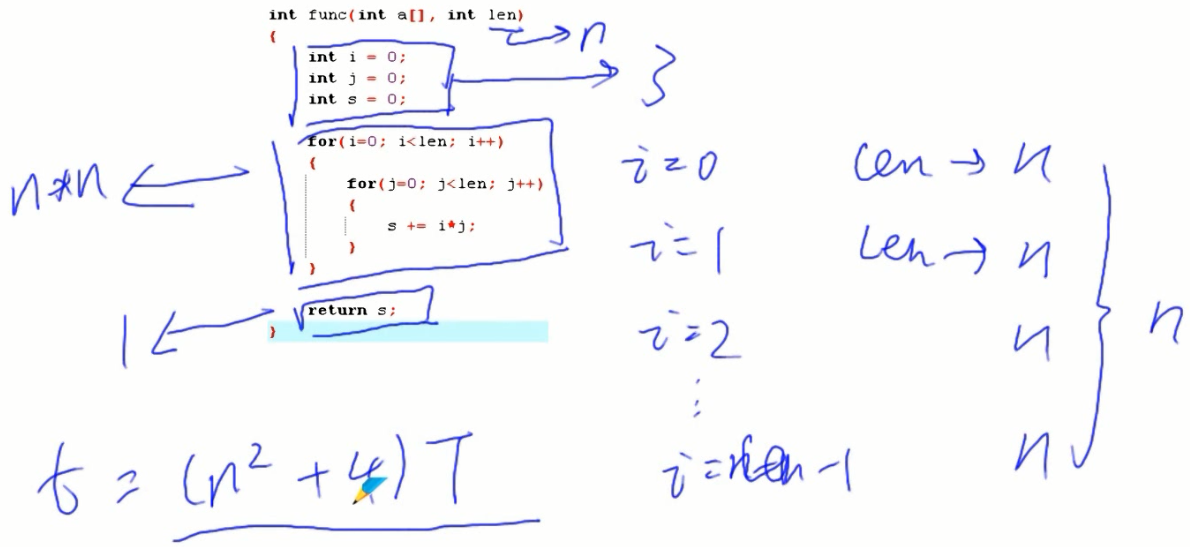
Sum1 t1 = （2n+5）\* T

Sum2 t2 = （n+3）\* T

Sum3 t3 = 3 \* T

t3 < t2 < t1 ，从理论的角度进行了分析。t3是最小的，是最好的。

**练习**



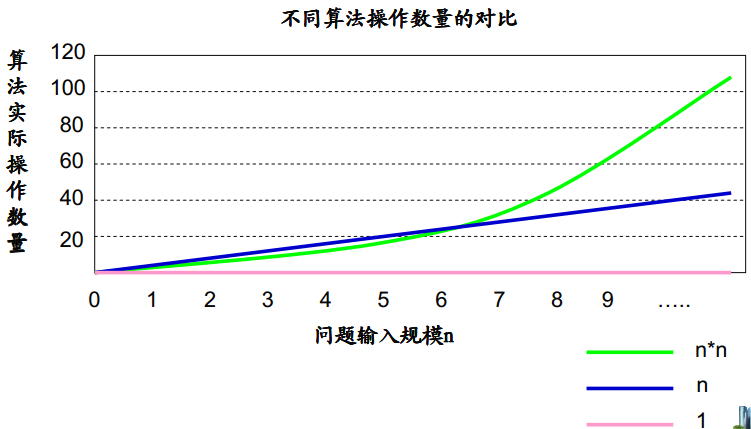
### 5）启示

练习中的程序关键部分的操作数量为n\*n

三种求和算法中求和的**关键部分的操作数量**分别为2n, n和1

* 随着问题规模n的增大，它们操作数量的差异会越来越大，因此实际算法在时间效率上的差异也会变得非常明显！

统计：





* **判断一个算法的效率时，往往只需要关注操作数量的最高次项，其它次要项和常数项可以忽略。**

### 6）大O表示法

* 算法效率严重依赖于操作(Operation)数量
* 在判断时首先关注操作数量的最高次项
* 操作数量的估算可以作为时间复杂度的估算

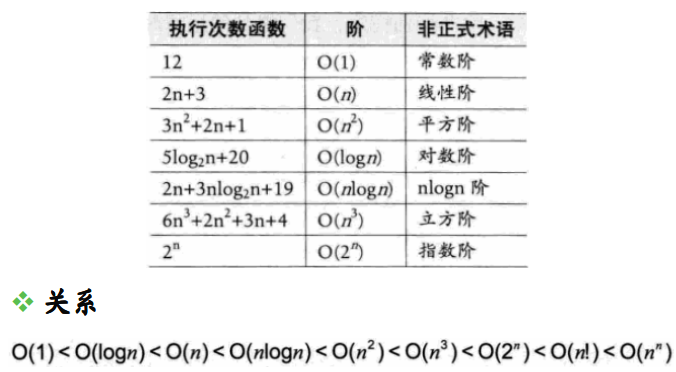
O(5) = O(1)

O(2n + 1) = O(2n) = O(n)

O(n2 + n + 1) = O(n2)

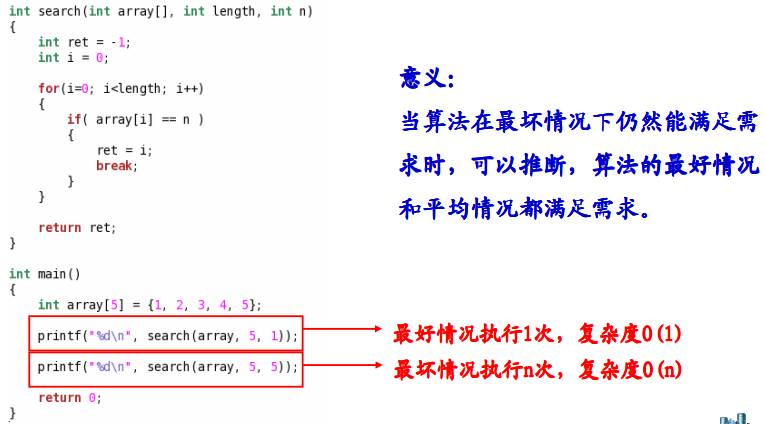
O(3n3+1) = O(3n3) = O(n3)

### 7）常见时间复杂度类型



如果想深入这些算法公式怎么推导出来的，课程《算法分析与设计》。

## 最坏与最好

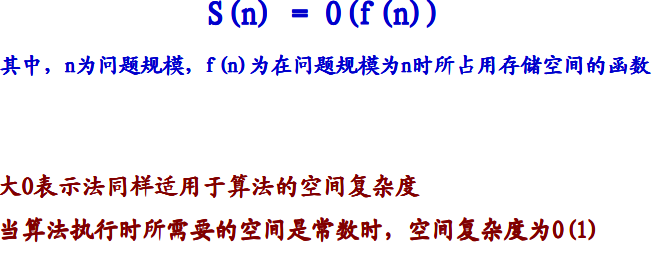


* **在没有特殊说明时，我们所分析的算法的时间复杂度都是指最坏时间复杂度。**

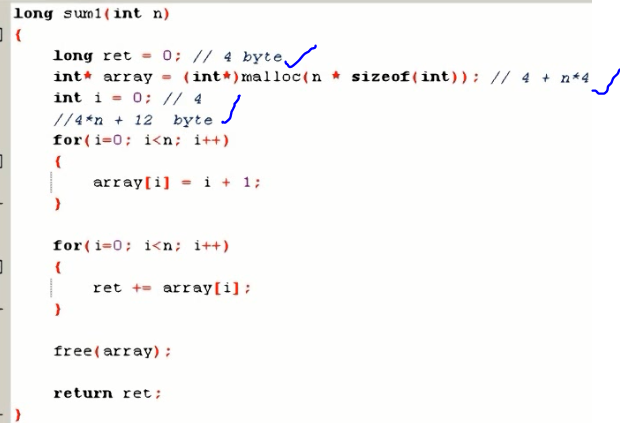
最坏的情况都能满足，最好的情况肯定能满足，因此我们在分析算法时，都要分析它的最坏情况，前面的大O表示法也是一样。

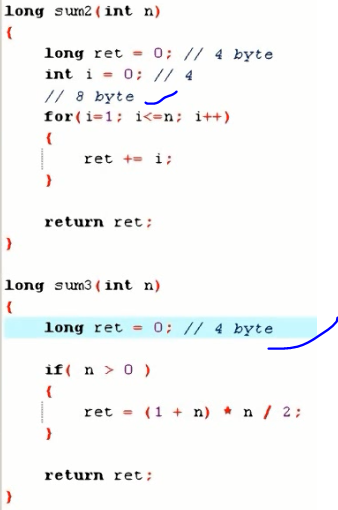
## 算法的空间复杂度

算法的空间复杂度通过计算算法的存储空间实现



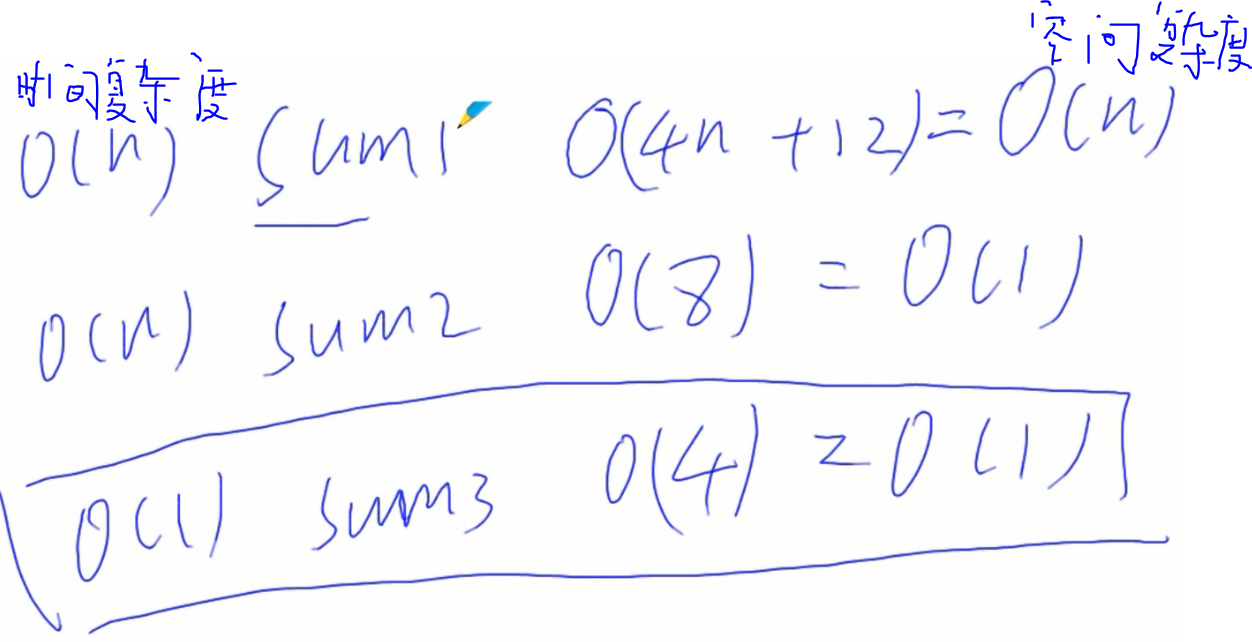
### 1）空间复杂度估算





**统计一下，三个算法分别占用了4n+12、8、4个字节。**

**用大O表示法转换，分别为：**



### 2）空间与时间的策略

1. 多数情况下，算法执行时所用的时间更令人关注
2. 如果有必要，可以通过增加空间复杂度来降低时间复杂度
3. 同理，也可以通过增加时间复杂度来降低空间复杂度

**在实现算法时需要分析具体问题对执行时间和空间的要求。**

要求快，空间换时间；要求小，时间换空间

eg： 空间换时间



若采用冒泡排序法等方法，大O肯定不是n，至少为n2。

### 3）课后思考

当两个算法的大O表示法相同的时候，是否意味着两个算法的效率完全相同？



# 五 线性表的本质

## 1 生活中的线性表

经典实例



## 2 线性表的定义

* 线性表(List)是零个或多个数据元素的集合
* 线性表中的数据元素之间是有顺序的
* 线性表中的数据元素个数是有限的
* 线性表中的数据元素的类型必须相同



**定义**

线性表是具有相同类型的 n（ ≥0）个数据元素的有限序列

（a1, a2, …, an）

ai 是表项，n 是表长度

## 3 线性表的性质

**性质**

* a0为线性表的第一个元素，只有一个后继
* an为线性表的最后一个元素，只有一个前驱
* 除a0和an外的其它元素ai，既有前驱，又有后继
* 线性表能够逐项访问和顺序存取

## 4 线性表的深度理解

下面的关系中可以用线性表描述的是 D

A. 班级中同学的友谊关系

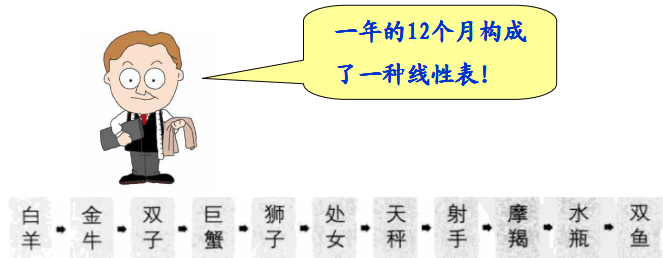
B. 公司中的上下级关系

C. 冬天图书馆排队占座关系

D. 花名册上名字之间的关系

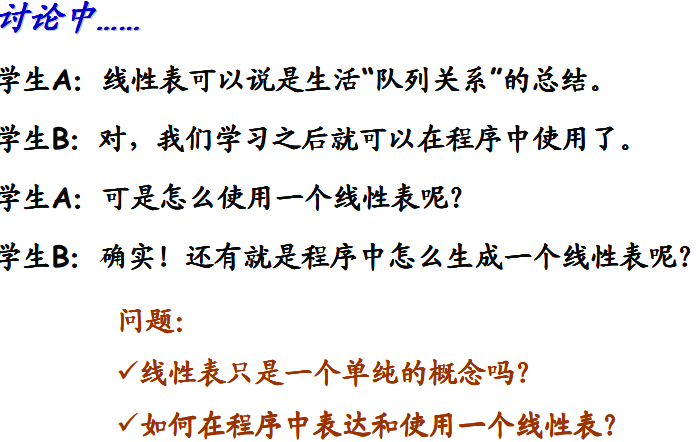
## 5 小结

* 线性表是数据元素的有序并且有限的集合
* 线性表中的数据元素必须是类型相同的
* 线性表可用于描述“**队列类型**”关系的问题



# 六 线性表的相关操作

## 1 线性表的讨论



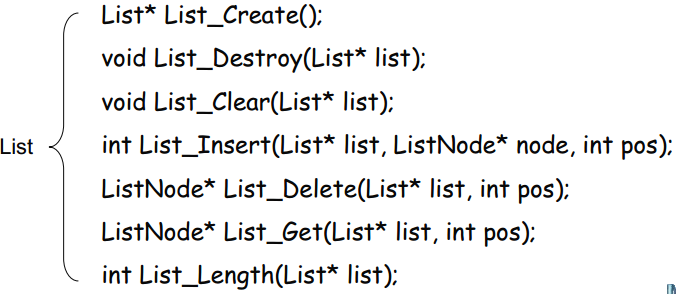
## 2 线性表的操作

**线性表的一些常用操作**

* + 创建线性表
  + 销毁线性表
  + 清空线性表
  + 将元素插入线性表
  + 将元素从线性表中删除
  + 获取线性表中某个位置的元素
  + 获取线性表的长度

## 3 线性表操作的实现

* 线性表在程序中表现为一种特殊的数据类型
* 线性表的操作在程序中的表现为一组函数



## 4 小结

* 线性表在程序中表现为一种特殊的数据类型
* 线性表的操作则表现为一组相关的函数

问题：

1.1. 线性表的各个函数如何实现呢？

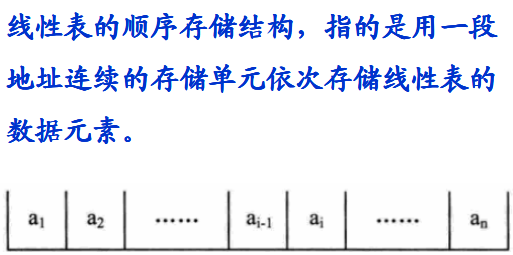
2.2. 有几种线性表的实现方式呢？

3.3. 每种实现方式的优缺点是什么呢？

# 七 线性表的顺序存储结构

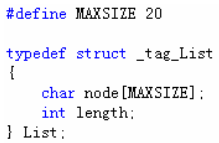
## 1 顺序存储定义

**地址连续的存储单元**



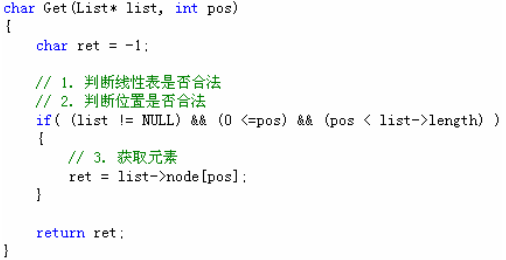
## 2 在C语言中可以用一维数组来实现顺序存储结构

* 存储空间的起始位置：数组node
* 线性表的最大容量：数组长度MAXSIZE
* 线性表的当前长度：length

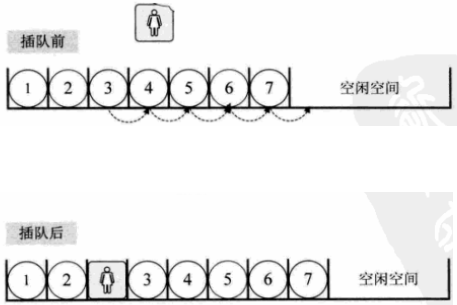


## 3 获取元素操作

* 判断线性表是否合法
* 判断位置是否合法
* 直接通过数组下标的方式获取元素

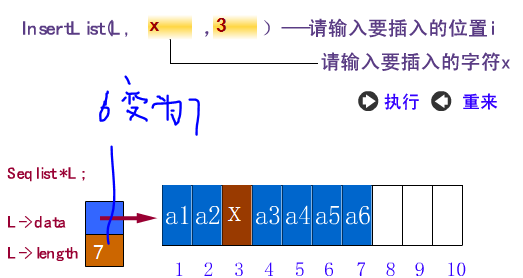


## 插入元素操作

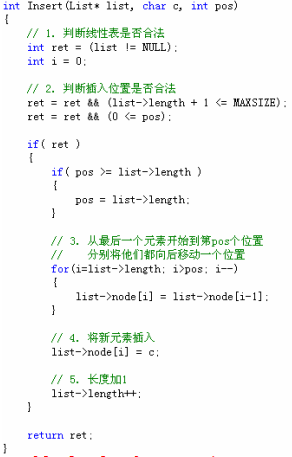


## 插入元素算法

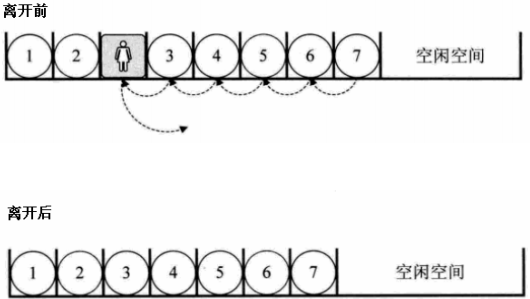
* 判断线性表是否合法
* 判断插入位置是否合法
* 把最后一个元素到插入位置的元素后移一个位置（整体移动，见动画）
* 将新元素插入
* 线性表长度加1



**代码的实现**

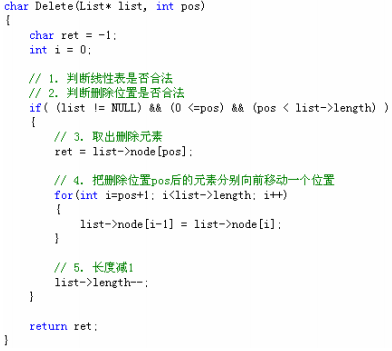


## 删除元素操作



## 7 删除元素算法

* 判断线性表是否合法
* 判断删除位置是否合法
* 将元素取出
* 将删除位置后的元素分别向前移动一个位置
* 线性表长度减1



## 8 小结

* 优点：

1）无需为线性表中的逻辑关系增加额外的空间

2）可以快速的获取表中合法位置的元素

* 缺点：

1）插入和删除操作需要移动大量元素

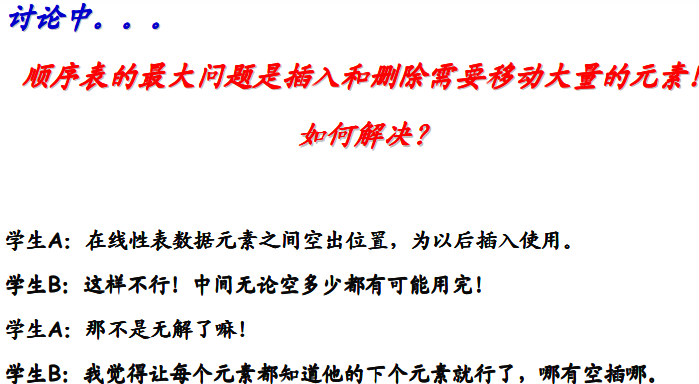
2）当线性表长度变化较大时难以确定存储空间的容量

## 9 个人总结

线性表的顺序结构就是：表头信息（表长、容量）+ 地址连续的空间来储存数据元素。

# 八 线性表的链式存储结构

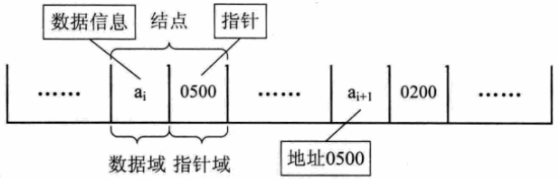
## 1 顺序表的思考



## 2 链式存储结构

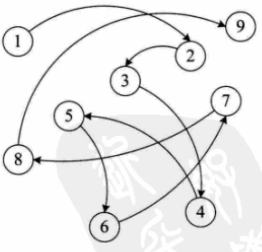
### 1）链式存储定义

* 为了表示每个数据元素与其直接后继元素之间的逻辑关系，每个元素除了存储本身的信息外，还需要存储指示其直接后继的信息。



### 2）链式存储逻辑结构

* n个结点链接成一个链式线性表的结构叫做**链表**，当每个结点中只包含一个指针域时，叫做**单链表**。



### 3）链表的基本概念

* **表头结点**

链表中的第一个结点，包含指向第一个数据元素的指针以及链表自身的一些信息。

* **数据结点**

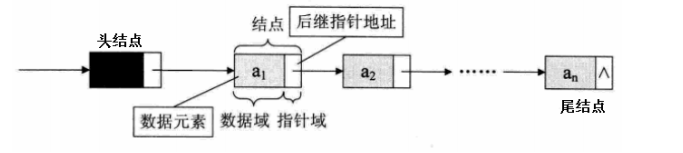
链表中代表数据元素的结点，包含指向下一个数据元素的指针和数据元素的信息。

* **尾结点**

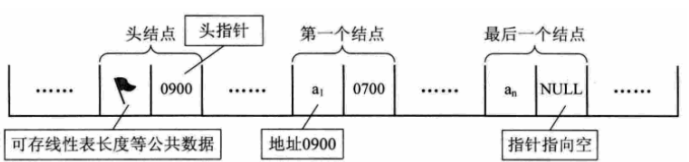
链表中的最后一个数据结点，其下一元素指针为空，表示无后继。

表头节点为第一个节点，但不是一个数据元素，作用：1.、它的指针域是指向第一个数据元素（白色部分）；2、包含整个链表的相关信息（黑色部分），比如包含整个链表的长度是多少。

* **单链表示例**



单链表：**逻辑导向**



单链表：**在物理内存中**

### 4）C语言链式存储结构