根据线性表的实际存储方式，分为两种实现模型：

顺序表 :将元素顺序地存放在一块连续的存储区里，元素间的顺序关系由它们的存储顺序自然表示。

链表 :将元素存放在通过链接构造起来的一系列存储块中

内存， 类型本质， 连续储存

内存是：以字节Byte为基本单位（1个字节有8位）连续的存储空间

普通整型Int：需要4个字节（对于32位机器）

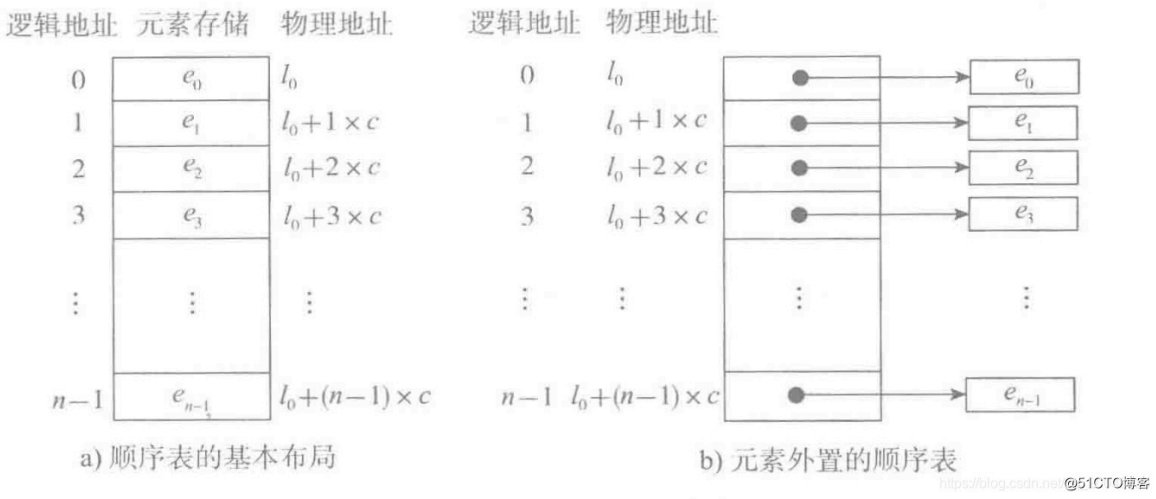
例子： Int a = 1: 二进制 0000 0001 （只有一个字节），但是Int 占4个字节，所以还有三个字节是0000 0000

字符串Char： 需要1个字节

所以：变量类型不同，需要的存储空间不同；计算机对二进制数据的方式不同

一组相同类型数据最简单的存储方式： 顺序表连续存储

顺序表的基本形式



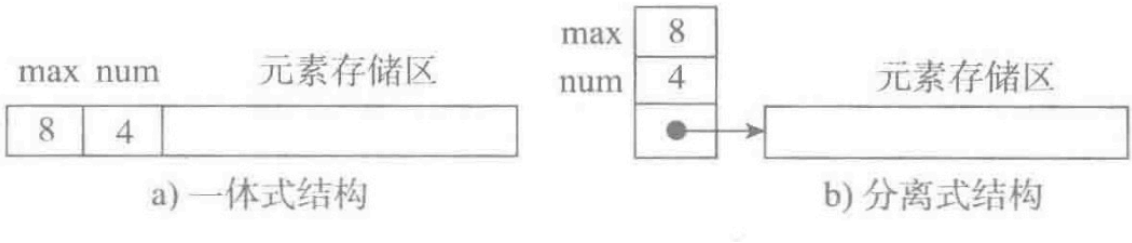
基本顺序表只能处理相同类型

元素外置的顺序表可以存储不同类型的数据，因为物理地址所占的大小是一样的（连续空间储存地址）

顺序表的结构与实现

一个顺序表的完整信息包括两部分，一部分是为实现正确操作而需记录的信息，即有关表的整体情况的信息，这部分信息主要包括元素存储区的**容量**和当前表中已有的**元素个数**两项；另一部分是表中的元素集合

顺序表的两种基本实现方式



- 图a为一体式结构，存储表信息的单元与元素存储区以连续的方式安排在一块存储区里，两部分数据的整体形成一个完整的顺序表对象。
- 图b为分离式结构，表对象里只保存与整个表有关的信息（即容量和元素个数），实际数据元素存放在另一个独立的元素存储区里，通过链接与基本表对象关联。

一体式结构整体性强，易于管理。但是由于数据元素存储区域是表对象的一部分，顺序表创建后，元素存储区就固定了。

元素存储区替换

一体式结构由于顺序表信息区与数据区连续存储在一起，所以若想更换数据区，则只能整体搬迁，即整个顺序表对象（指存储顺序表的结构信息的区域）改变了。

分离式结构若想更换数据区，只需将表信息区中的数据区链接地址更新即可，而该顺序表对象不变。

元素存储区扩充

可以扩充的分离式结构顺序表： **动态顺序表**

扩充策略：

- 线性增长：每次扩充固定树木的存储位置

  节省空间，操作频繁
- 加倍增长： 每次扩充容量加倍

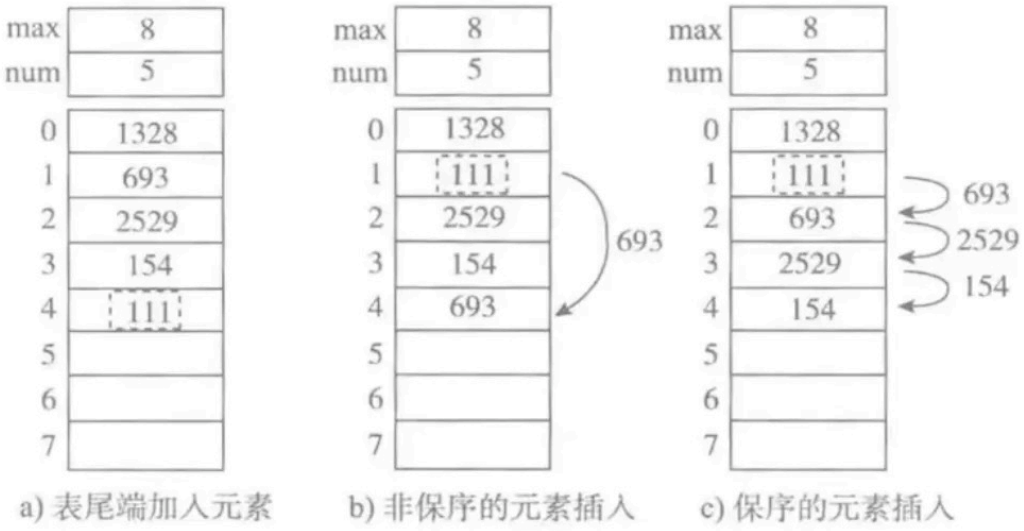
  减少操作次数， 可能浪费空间资源，以空间换时间（推荐）

顺序表的操作

增加元素

增加元素

如图所示，为顺序表增加新元素111的三种方式



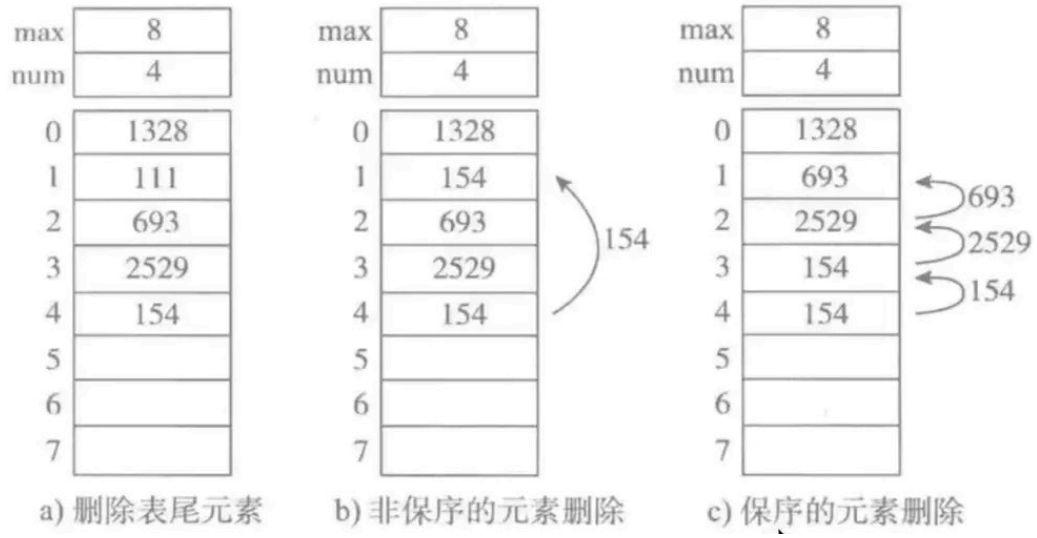
a. 尾端加入元素，时间复杂度为O(1)

b. 非保序（顺序可变）的加入元素（不常见），时间复杂度为O(1)

c. 保序（顺序不可变）的元素加入，时间复杂度为O(n)

删除元素

删除元素



a. 删除表尾元素，时间复杂度为O(1)

b. 非保序的元素删除（不常见），时间复杂度为O(1)

c. 保序的元素删除，时间复杂度为O(n)

Python中的顺序表

Python中的list和tuple两种类型采用了顺序表的实现技术，具有前面讨论的顺序表的所有性质。

tuple不可以修改，其他与list的性质类似。

list的基本实现技术

基于下标（位置）的高效元素访问和更新，时间复杂度应该是O(1)；

为满足该特征，应该采用顺序表技术，表中元素保存在一块连续的存储区中。

允许任意加入元素，而且在不断加入元素的过程中，表对象的标识（函数id得到的值）不变

为满足该特征，就必须能更换元素存储区，并且为保证更换存储区时list对象的标识id不变，只能采用分离式实现技术。

在Python的官方实现中，**list就是一种采用分离式技术实现的动态顺序表**。这就是为什么用list.append(x) （或 list.insert(len(list), x)，即尾部插入）比在指定位置插入元素效率高的原因。

```
1
```