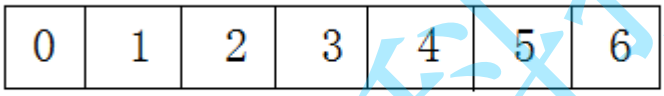
Lesson3--顺序表和链表

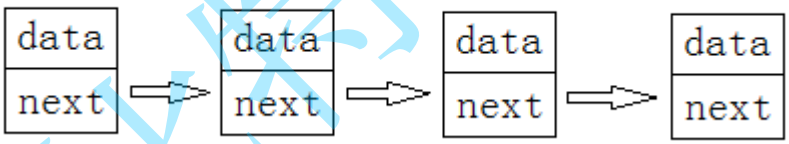
【本节目标】

- 1.线性表
- 2.顺序表
- 3.链表
- 4.顺序表和链表的区别和联系

1.线性表

> 线性表 *(linear list)* 是n个具有相同特性的数据元素的有限序列。 线性表是一种在实际中广泛使用的数据结构，常见的线性表：顺序表、链表、栈、队列、字符串...
>
> 线性表在逻辑上是线性结构，也就说是连续的一条直线。但是在物理结构上并不一定是连续的，线性表在物理上存储时，通常以数组和链式结构的形式存储。





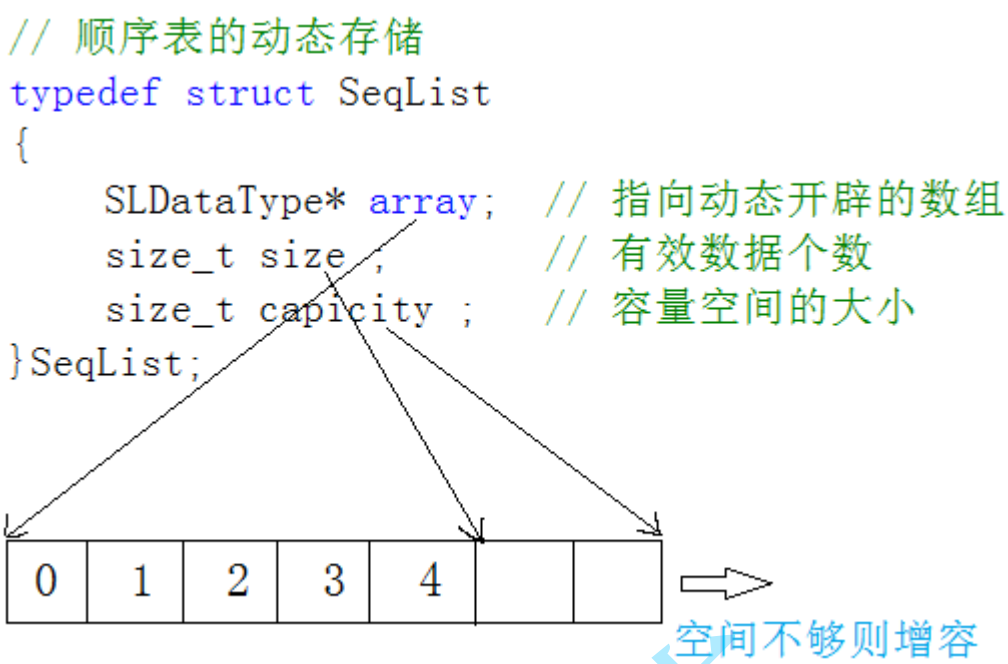
2.顺序表

> ### 2.1概念及结构
>
> 顺序表是用一段**物理地址连续**的存储单元依次存储数据元素的线性结构，一般情况下采用数组存储。在数组上完成数据的增删查改。
>
> 顺序表一般可以分为：
>
> 1. 静态顺序表：使用定长数组存储。
> 2. 动态顺序表：使用动态开辟的数组存储。

```
// 顺序表的静态存储
#define N 100
typedef int SLDataType;

typedef struct SeqList
{
     SLDataType array[N]; // 定长数组
```

```
    size_t size;        // 有效数据的个数
}SeqList;


// 顺序表的动态存储
typedef struct SeqList
{
    SLDataType* array;  // 指向动态开辟的数组
    size_t size ;       // 有效数据个数
    size_t capicity ;   // 容量空间的大小
}SeqList;
```

```
// 顺序表的静态存储
#define N 7
typedef int SLDataType;

typedef struct SeqList
{
    SLDataType array[N];    // 定长数组
    size_t size;            // 有效数据的个数
}SeqList;
```



2.2 接口实现：

静态顺序表只适用于确定知道需要存多少数据的场景。静态顺序表的定长数组导致N定大了，空间开多了浪费，开少了不够用。所以现实中基本都是使用动态顺序表，根据需要动态的分配空间大小，所以下面我们实现动态顺序表。

```
// 顺序表的动态存储
typedef struct SeqList
{
     SLDataType* array;  // 指向动态开辟的数组
     size_t size ;       // 有效数据个数
     size_t capicity ;   // 容量空间的大小
}SeqList;

// 基本增删查改接口
// 顺序表初始化
void SeqListInit(SeqList* psl, size_t capacity);
// 顺序表销毁
void SeqListDestory(SeqList* psl);
// 顺序表打印
void SeqListPrint(SeqList* psl);
// 检查空间，如果满了，进行增容
void CheckCapacity(SeqList* psl);
// 顺序表尾插
void SeqListPushBack(SeqList* psl, SLDataType x);
// 顺序表尾删
void SeqListPopBack(SeqList* psl);
// 顺序表头插
void SeqListPushFront(SeqList* psl, SLDataType x);
// 顺序表头删
void SeqListPopFront(SeqList* psl);
// 顺序表查找

int SeqListFind(SeqList* psl, SLDataType x);
```

```
// 顺序表在pos位置插入x
void SeqListInsert(SeqList* psl, size_t pos, SLDataType x);
// 顺序表删除pos位置的值
void SeqListErase(SeqList* psl, size_t pos);
```

2.3 数组相关面试题

1. 原地移除数组中所有的元素val，要求时间复杂度为O(N)，空间复杂度为O(1)。OJ链接
2. 删除排序数组中的重复项。OJ链接
3. 合并两个有序数组。OJ链接
4. 旋转数组。OJ链接
5. 数组形式的整数加法。OJ链接

2.4 顺序表的问题及思考

问题：

1. 中间/头部的插入删除，时间复杂度为O(N)
2. 增容需要申请新空间，拷贝数据，释放旧空间。会有不小的消耗。
3. 增容一般是呈2倍的增长，势必会有一定的空间浪费。例如当前容量为100，满了以后增容到200，我们再继续插入了5个数据，后面没有数据插入了，那么就浪费了95个数据空间。

思考：

如何解决以上问题呢？下面给出了链表的结构来看看。

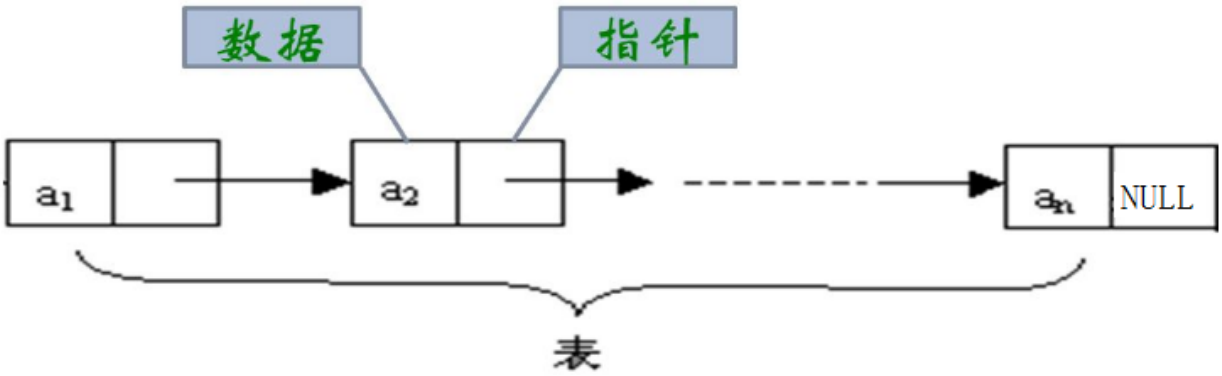
3.链表

3.1 链表的概念及结构

概念：链表是一种**物理存储结构上非连续**、非顺序的存储结构，数据元素的**逻辑顺序**是通过链表中的**指针链接**次序实现的 。

现实中：

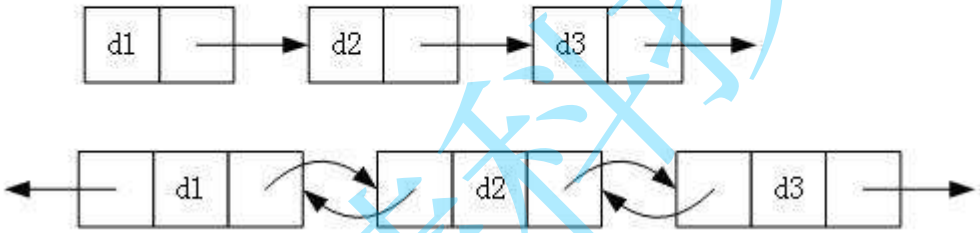
数据结构中：



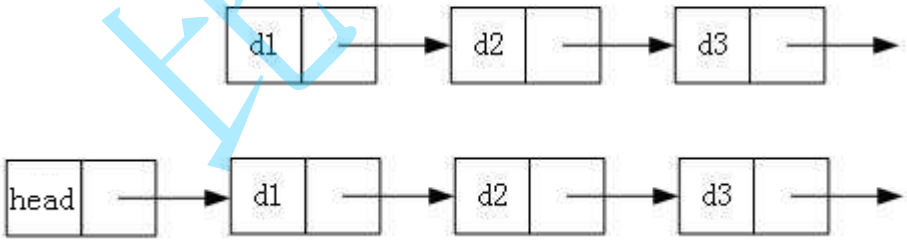
实际中链表的结构非常多样，以下情况组合起来就有8种链表结构：

1. 单向、双向
2. 带头、不带头
3. 循环、非循环

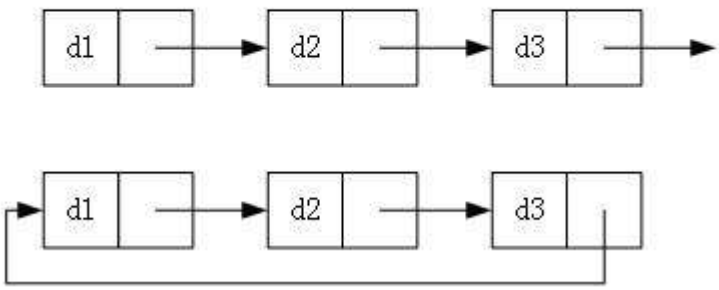
1. 单链表、双向链表



2. 不带头单链表、带头链表

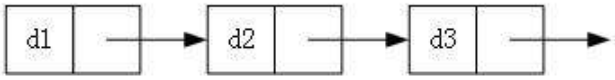


3. 单链表、循环单链表

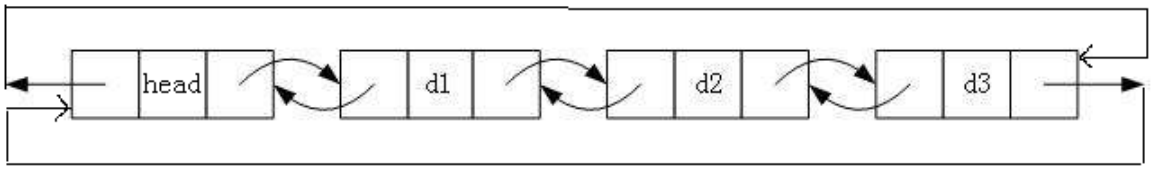


虽然有这么多的链表的结构，但是我们实际中最常用还是两种结构：

无头单向非循环链表



带头双向循环链表



1. 无头单向非循环链表：**结构简单**，一般不会单独用来存数据。实际中更多是作为**其他数据结构的子结构**，如哈希桶、图的邻接表等等。另外这种结构在**笔试面试**中出现很多。
2. 带头双向循环链表：**结构最复杂**，一般用在单独存储数据。实际中使用的链表数据结构，都是带头双向循环链表。另外这个结构虽然结构复杂，但是使用代码实现以后会发现结构会带来很多优势，实现反而简单了，后面我们代码实现了就知道了。

3.2链表的实现

```
// 1、无头+单向+非循环链表增删查改实现
typedef int SLTDateType;
typedef struct SListNode
{
    SLTDateType data;
    struct SListNode* next;
}SListNode;

// 动态申请一个节点
SListNode* BuySListNode(SLTDateType x);
// 单链表打印
void SListPrint(SListNode* plist);
// 单链表尾插
void SListPushBack(SListNode** pplist, SLTDateType x);
// 单链表的头插
void SListPushFront(SListNode** pplist, SLTDateType x);
// 单链表的尾删
void SListPopBack(SListNode** pplist);
// 单链表头删
void SListPopFront(SListNode** pplist);
// 单链表查找
SListNode* SListFind(SListNode* plist, SLTDateType x);
// 单链表在pos位置之后插入x
// 分析思考为什么不在pos位置之前插入?
void SListInsertAfter(SListNode* pos, SLTDateType x);
// 单链表删除pos位置之后的值

// 分析思考为什么不删除pos位置?
```

```
void SListEraseAfter(SListNode* pos);
```

3.3 链表面试题

1. 删除链表中等于给定值 **val** 的所有节点。 OJ链接
2. 反转一个单链表。 OJ链接
3. 给定一个带有头结点 head 的非空单链表，返回链表的中间结点。如果有两个中间结点，则返回第二个中间结点。OJ链接
4. 输入一个链表，输出该链表中倒数第k个结点。 OJ链接
5. 将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。OJ链接
6. 编写代码，以给定值x为基准将链表分割成两部分，所有小于x的结点排在大于或等于x的结点之前 。OJ链接
7. 链表的回文结构。OJ链接
8. 输入两个链表，找出它们的第一个公共结点。OJ链接
9. 给定一个链表，判断链表中是否有环。 OJ链接
10. 给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 NULL OJ链接
11. 给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。要求返回这个链表的深度拷贝。OJ链接
12. 对链表进行插入排序。OJ链表
13. 在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 OJ链接
14. 其他 。ps：链表的题当前因为难度及知识面等等原因还不适合我们当前学习，以后大家自己下去以后 Leetcode OJ链接 + 牛客 OJ链接

```
// 2、带头+双向+循环链表增删查改实现
typedef int LTDataType;
typedef struct ListNode
{
    LTDataType _data;
    struct ListNode* _next;
    struct ListNode* _prev;
}ListNode;

// 创建返回链表的头结点.
ListNode* ListCreate();
// 双向链表销毁
void ListDestory(ListNode* plist);
// 双向链表打印
void ListPrint(ListNode* plist);
// 双向链表尾插
void ListPushBack(ListNode* plist, LTDataType x);
// 双向链表尾删
void ListPopBack(ListNode* plist);

// 双向链表头插
```

```
void ListPushFront(ListNode* plist, LTDataType x);
// 双向链表头删
void ListPopFront(ListNode* plist);
// 双向链表查找
ListNode* ListFind(ListNode* plist, LTDataType x);
// 双向链表在pos的前面进行插入
void ListInsert(ListNode* pos, LTDataType x);
// 双向链表删除pos位置的节点
void ListErase(ListNode* pos);
```

4.顺序表和链表的区别和联系

顺序表：一白遮百丑

白：空间连续、支持随机访问

丑：1.中间或前面部分的插入删除时间复杂度O(N) 2.增容的代价比较大。


链表：一(黑)毁所有

黑：以节点为单位存储，不支持随机访问

所有：1.任意位置插入删除时间复杂度为O(1) 2.没有增容问题，插入一个开辟一个空间。

5.顺序表和链表概念选择题

概念选择题：

1.在一个长度为n的顺序表中删除第i个元素，要移动_____个元素。如果要在第i个元素前插入一个元素，要后移_____个元素。
A n-i, n-i+1
B n-i+1, n-i
C n-i, n-i
D n-i+1, n-i+1

2.取顺序表的第i个元素的时间同i的大小有关()
A 对
B 错

3.在一个具有 n 个结点的有序单链表中插入一个新结点并仍然保持有序的时间复杂度是 。
A O（1）
B O（n）
C O（n2）
D O（nlog2n）

4.下列关于线性链表的叙述中，正确的是（ ）。
A 各数据结点的存储空间可以不连续，但它们的存储顺序与逻辑顺序必须一致
B 各数据结点的存储顺序与逻辑顺序可以不一致，但它们的存储空间必须连续
C 进行插入与删除时，不需要移动表中的元素
D 以上说法均不正确

5.设一个链表最常用的操作是在末尾插入结点和删除尾结点，则选用（）最节省时间。
A 单链表
B 单循环链表
C 带尾指针的单循环链表
D 带头结点的双循环链表

6.链表不具有的特点是（）。
A 插入、删除不需要移动元素
B 不必事先估计存储空间
C 可随机访问任一元素
D 所需空间与线性表长度成正比

7.在一个单链表中，若删除 P 所指结点的后续结点，则执行?
A p = p->next; p->next = p->next->next;
B p->next = p->next;
C p->next = p->next->next;
D p = p->next->next

8.一个单向链表队列中有一个指针p，现要将指针r插入到p之后，该进行的操作是____。
A p->next=p->next->next
B r->next=p;p->next=r->next
C r->next=p->next;p->next=r
D r=p->next;p->next=r->next
E r->next=p;p->next=r
F p=p->next->next

答案:

1.A
2.B
3.B
4.C
5.D
6.C
7.C
8.C