顺序表（顺序存储结构）的特点

（1）利用数据元素的存储位置表示线性表中相邻数据元素之间的前后关系，即线性表的**逻辑结构与存储结构一致**

（2）在访问线性表时，可以快速地计算出任何一个数据元素的存储地址。因此可以粗略地认为，**访问每个元素所花时间相等**

这种存取元素的方法被称为**随机存取法**

顺序表的优缺点

优点：

- ✓**存储密度大**（结点本身所占存储量/结点结构所占存储量）
- ✓可以**随机存取**表中任一元素

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

为克服这一缺点 → **链表**

2.5 线性表的链式表示和实现

链式存储结构

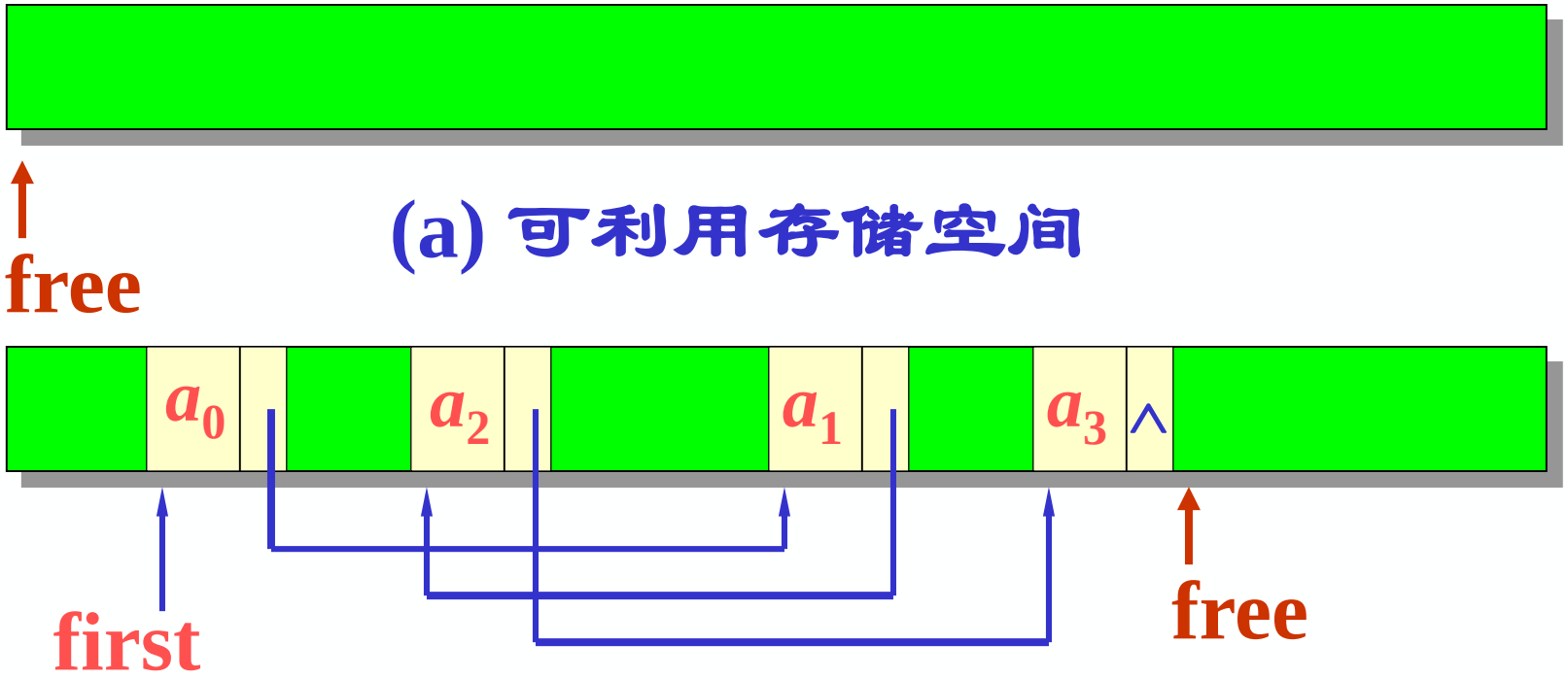
结点在存储器中的位置是任意的，即逻辑上相邻的数据元素在物理上不一定相邻

线性表的链式表示又称为非顺序映像或链式映像。

如何实现?

通过**指针**来实现

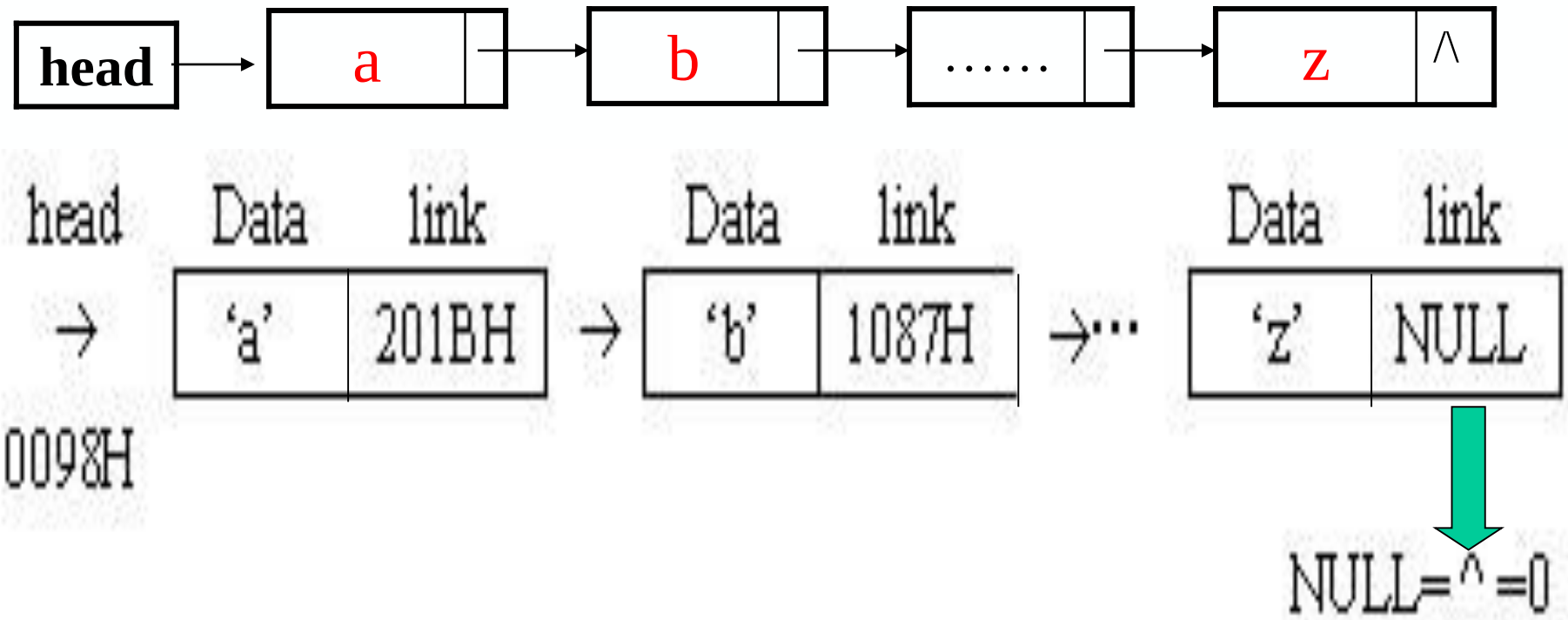
单链表的存储映像

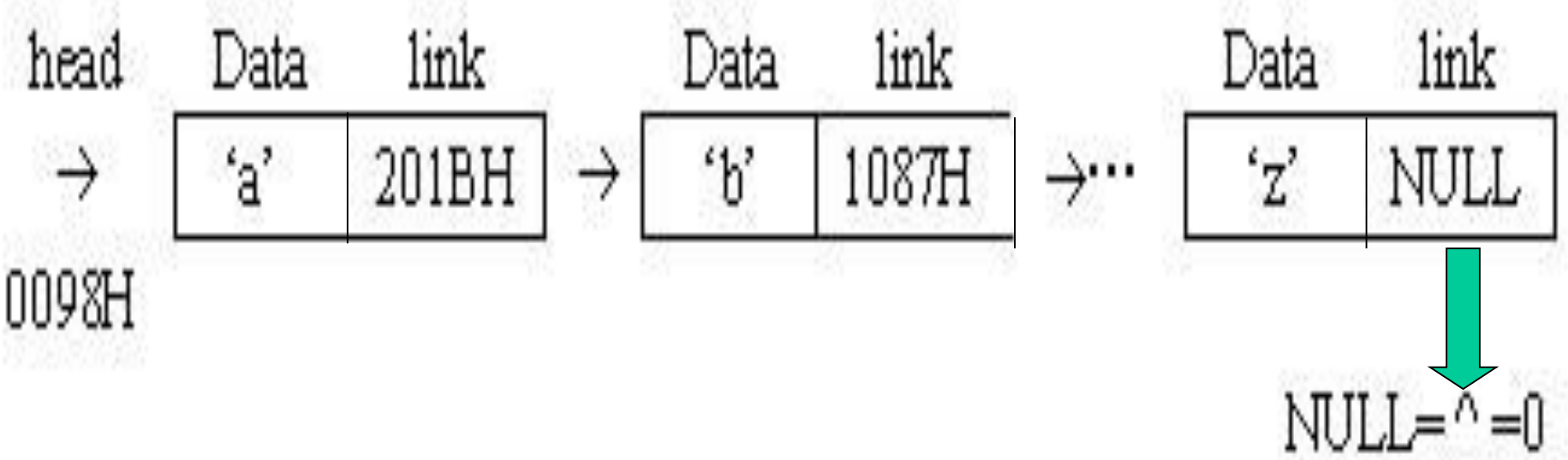


例　画出26 个英文字母表的链式存储结构

逻辑结构：　(a, b,　… ,y, z)

链式存储结构：





各结点由两个域组成：

数据	指针

数据域：存储元素数值数据

指针域：存储直接后继结点的存储位置

与链式存储有关的术语

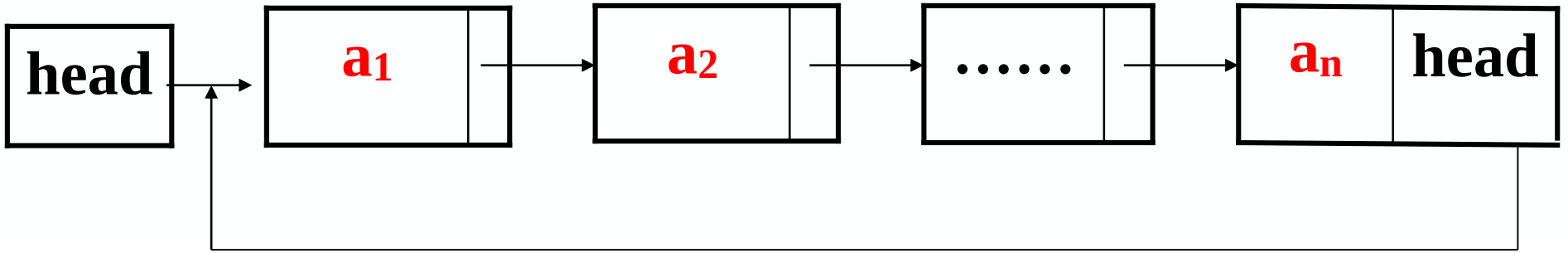
1、**结点：** 数据元素的存储映像。由数据域和指针域两部分组成

2、**链表：** n 个结点由**指针链**组成一个链表。它是线性表的链式存储映像，称为线性表的链式存储结构

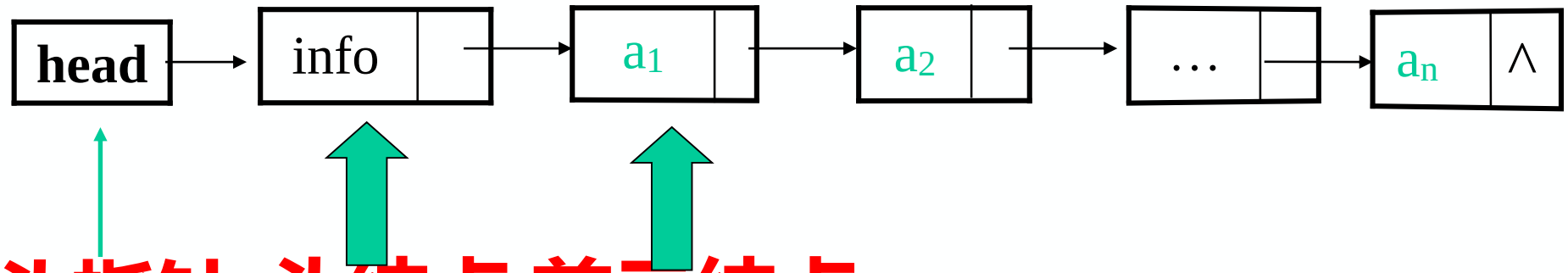
3、单链表、双链表、循环链表：

- 结点只有一个指针域的链表，称为**单链表**或**线性链表**
- 有两个指针域的链表，称为**双链表**
- 首尾相接的链表称为**循环链表**

循环链表示意图：



4、头指针、头结点和首元结点

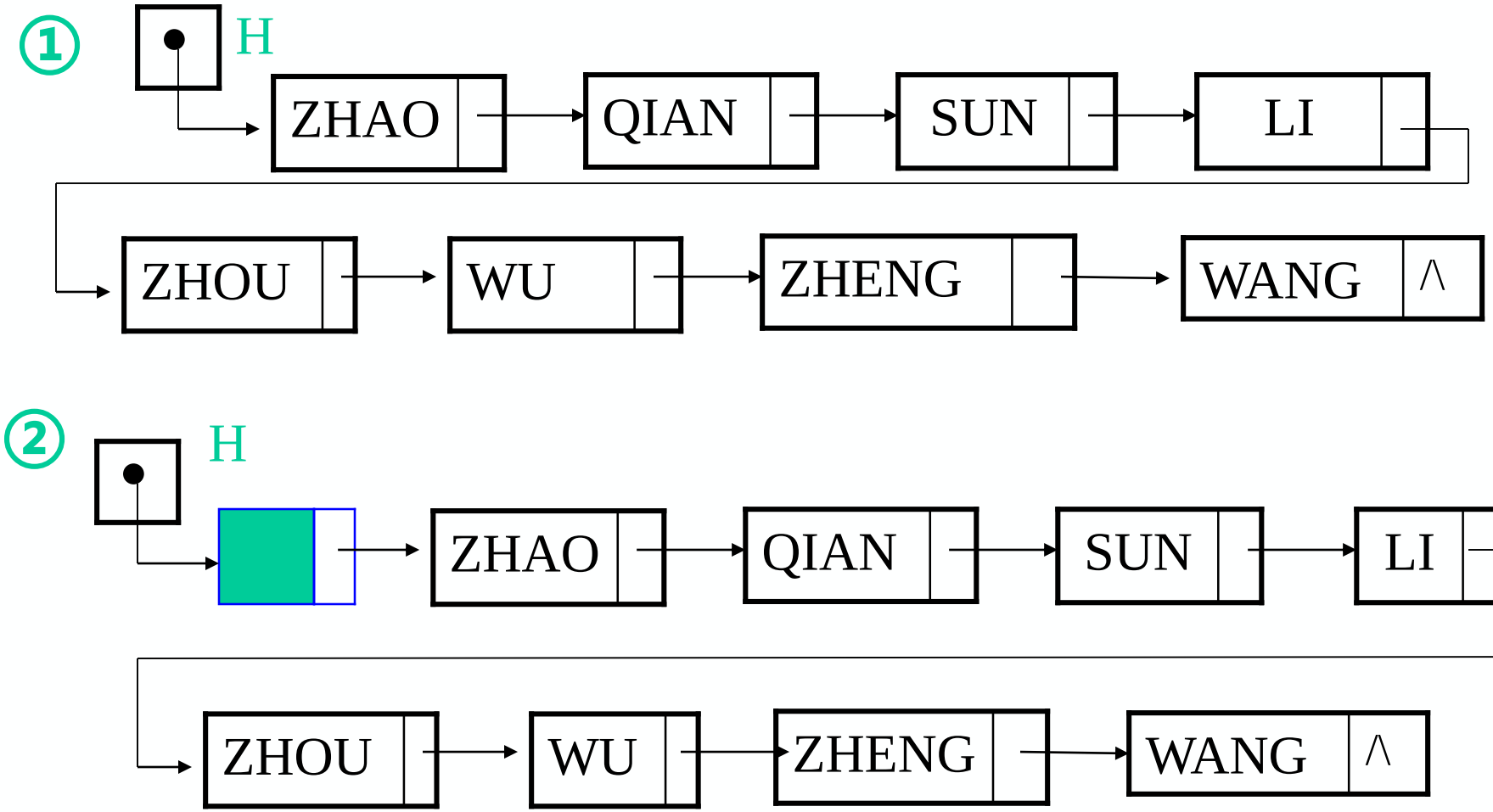


头指针是指向链表中第一个结点的指针

首元结点是指链表中存储第一个数据元素a_1的结点

头结点是在链表的首元结点之前附设的一个结点；数据域内只放空表标志和表长等信息

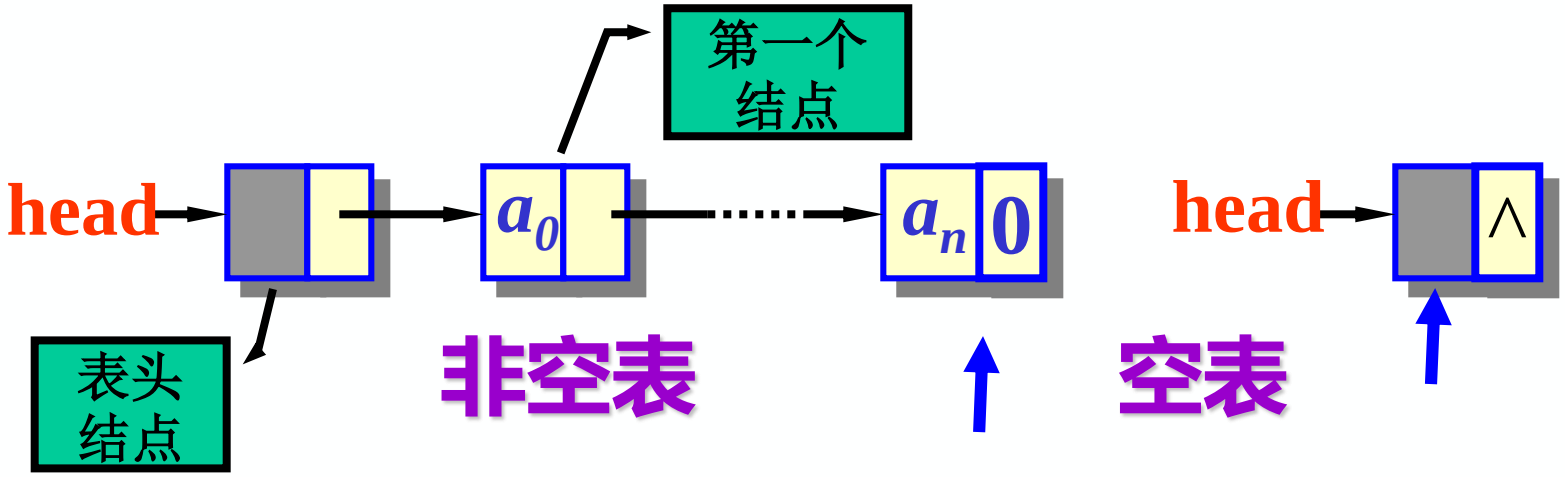
上例链表的逻辑结构示意图有以下两种形式：



区别：① 无头结点　② 有头结点

讨论1. 如何表示空表?

有头结点时，当头结点的指针域为空时表示空表



讨论2. 在链表中设置头结点有什么好处?

1 便于首元结点的处理

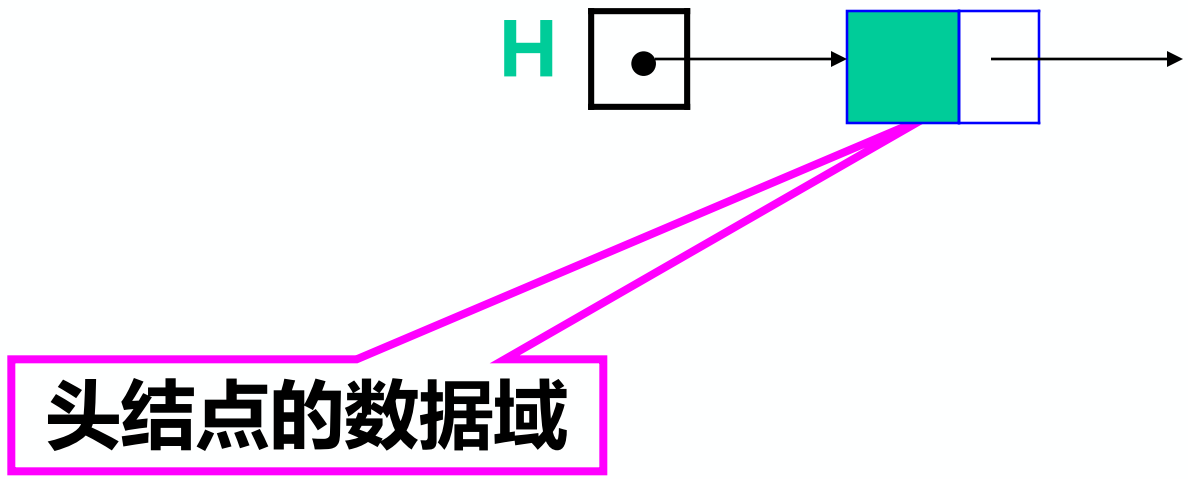
首元结点的地址保存在头结点的指针域中，所以在链表的第一个位置上的操作和其它位置一致，无须进行特殊处理;

2 便于空表和非空表的统一处理

无论链表是否为空，头指针都是指向头结点的非空指针，因此空表和非空表的处理也就统一了。

讨论3. 头结点的数据域内装的是什么?

头结点的数据域可以为空，也可存放线性表长度等附加信息，但此结点不能计入链表长度值。



链表（链式存储结构）的特点

（1）结点在存储器中的位置是任意的，即逻辑上相邻的数据元素在物理上不一定相邻

（2）访问时只能通过头指针进入链表，并通过每个结点的指针域向后扫描其余结点，所以寻找第一个结点和最后一个结点所花费的时间不等

这种存取元素的方法被称为**顺序存取法**

链表的优缺点

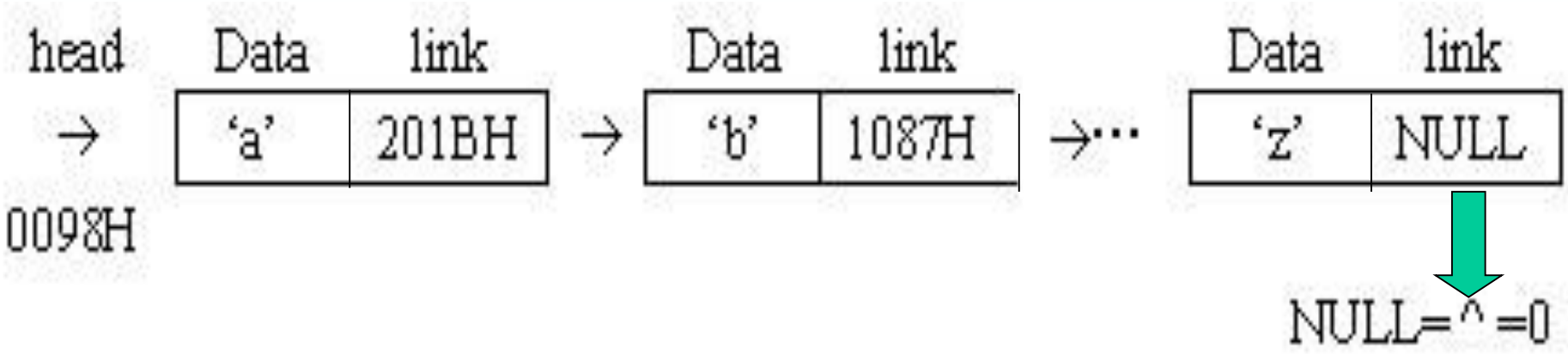
优点

- 数据元素的个数可以自由扩充
- 插入、删除等操作不必移动数据，只需修改链接指针，修改效率较高

链表的优缺点

缺点

- 存储密度小
- 存取效率不高，必须采用**顺序存取**，即存取数据元素时，只能按链表的顺序进行访问（顺藤摸瓜）



练习

1.链表的每个结点中都恰好包含一个指针。

2.顺序表结构适宜于进行顺序存取，而链表适宜于进行随机存取。

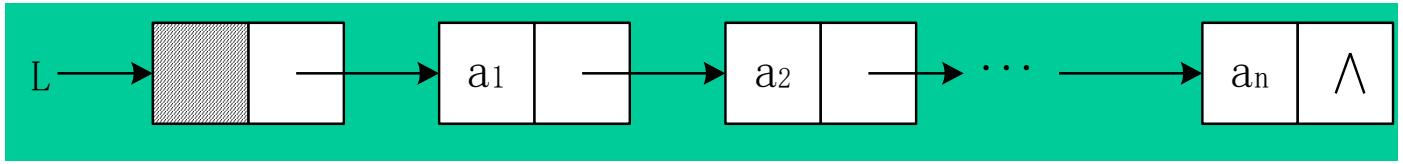
3.顺序存储方式的优点是存储密度大，且插入、删除运算效率高。

4.线性表若采用链式存储时，结点之间和结点内部的存储空间都是可以不连续的。

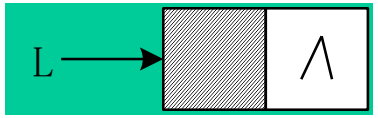
5.线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型

2.5.1 单链表的定义和实现

非空表



空表



- ✓ 单链表是由表头唯一确定，因此单链表可以用头指针的名字来命名
- ✓ 若头指针名是L，则把链表称为表L

单链表的存储结构定义

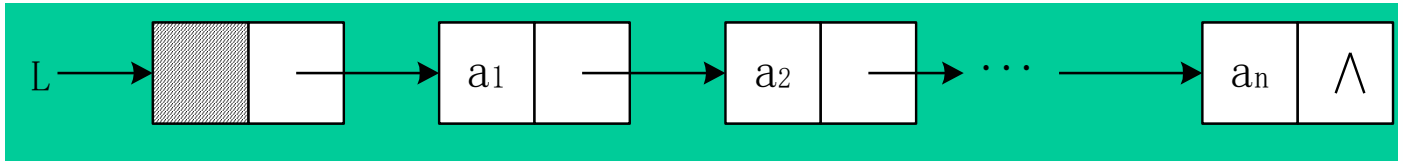
```
typedef struct LNode{
    ElemType   data;          //数据域
    struct LNode  *next;      //指针域
}LNode,*LinkList;
        // *LinkList为Lnode类型的指针
```

LNode *p ⟷ **LinkList p**

注意区分指针变量和结点变量两个不同的概念

- 指针变量p：表示结点地址
- 结点变量*p：表示一个结点

LNode *p



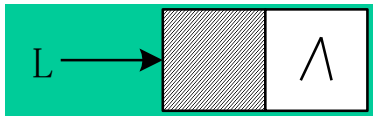
若p->data=a_i, 则p->next->data=a_{i+1}

16:15

2.5.2 单链表基本操作的实现

1. 初始化
2. 取值
3. 查找
4. 插入
5. 删除

1.初始化(构造一个空表）



【算法步骤】

（1）生成新结点作头结点，用头指针L指向头结点。

（2）头结点的指针域置空。

【算法描述】

```
Status InitList_L(LinkList &L){
   L=new LNode;
   L->next=NULL;
   return OK;
}
```

补充：几个简单基本操作的算法实现

销毁

```
Status DestroyList_L(LinkList &L){
    LinkList p;
      while(L)
       {
          p=L;
          L=L->next;
          delete p;
       }
     return OK;
 }
```

补充：几个简单基本操作的算法实现

清空

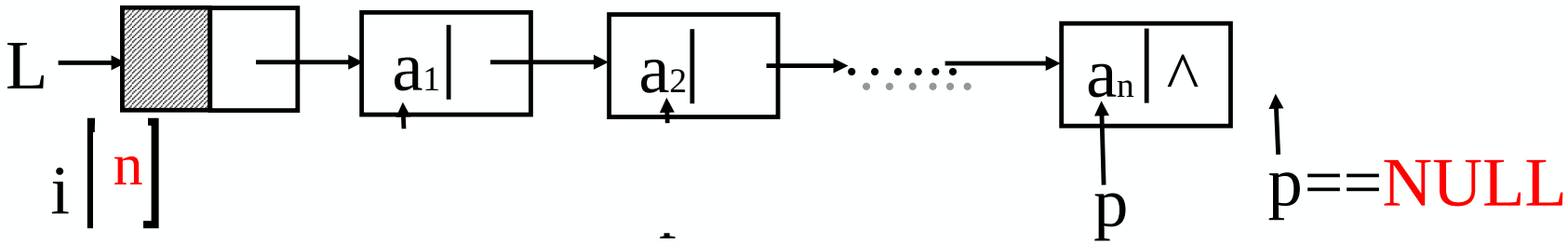
```
Status ClearList(LinkList & L){
  // 将L重置为空表
   LinkList p,q;
   p=L->next;   //p指向第一个结点
   while(p)       //没到表尾
     {  q=p->next; delete p;     p=q;   }
   L->next=NULL;   //头结点指针域为空
   return OK;
 }
```

补充：几个简单基本操作的算

求表长

“数”结点：

- 指针p依次指向各个结点
- 从第一个元素开始“数”
- 一直“数”到最后一个结点




```
p=L->next;
i=0;
while(p){i++;p=p->next;}
```

求表长

```
int  ListLength_L(LinkList L){
//返回L中数据元素个数
    LinkList p;
    p=L->next;  //p指向第一个结点
     i=0;
     while(p){//遍历单链表,统计结点数
           i++;
           p=p->next;    }
    return i;
 }
```

“数”结点：

- 指针p依次指向各个结点
- 从第一个元素开始“数”
- 一直“数”到最后一个结点

判断表是否为空

```
int ListEmpty(LinkList L){
//若L为空表，则返回1，否则返回0
  if(L->next)  //非空
    return 0;
  else
    return 1;
}
```

线性表的重要基本操作

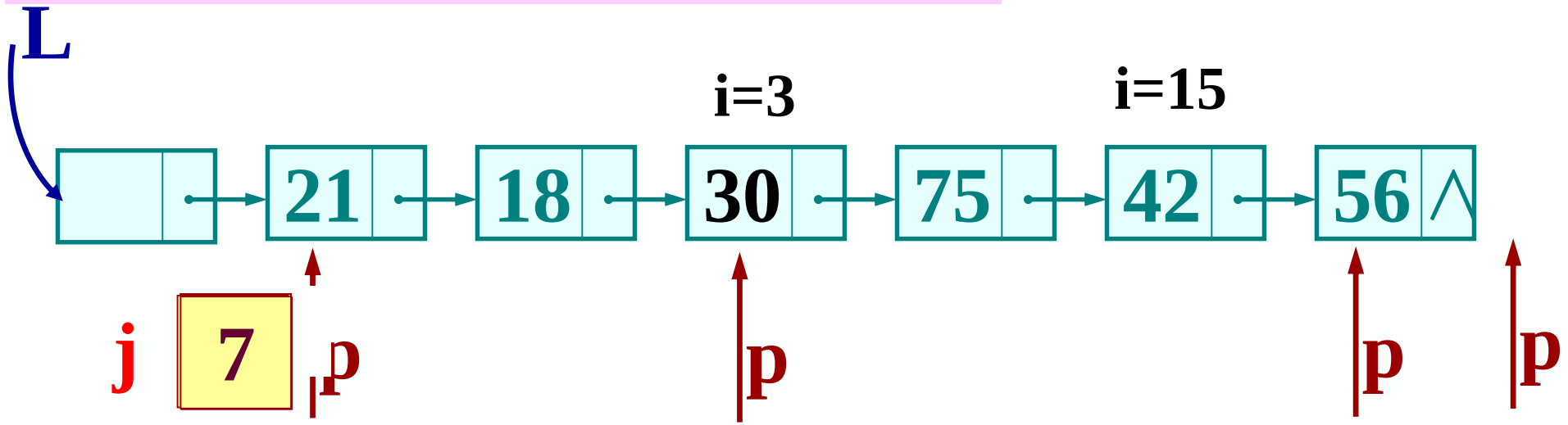
1. 初始化
2. 取值
3. 查找
4. 插入
5. 删除

2. 取值（根据位置i获取相应位置数据元素的内容）

- 思考：顺序表里如何找到第i个元素?

- 链表的查找：要从链表的头指针出发，顺着链域next逐个结点往下搜索，直至搜索到第i个结点为止。因此，链表不是随机存取结构

例：分别取出表中i=3和i=15的元素



【算法步骤】

✓从第1个结点（L->next）顺链扫描，用指针p指向当前扫描到的结点，p初值p = L->next。

✓*j*做计数器，累计当前扫描过的结点数，*j*初值为1。

✓当p指向扫描到的下一结点时，计数器*j*加1。

✓当*j* = *i*时，p所指的结点就是要找的第*i*个结点。

16:15

2. 取值（根据位置i获取相应位置数据元素的内容）

```
//获取线性表L中的某个数据元素的内容
Status GetElem_L(LinkList L,int i,ElemType &e){
    p=L->next;j=1; //初始化
     while(p&&j<i){    //向后扫描，直到p指向第i个元素或p为空
       p=p->next; ++j;
     }
     if(!p || j>i)return ERROR; //第i个元素不存在
     e=p->data; //取第i个元素
     return OK;
}//GetElem_L
```