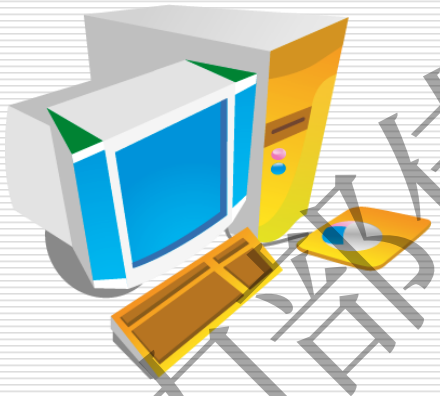
# 数据结构



## 第二章　线性表

人工智能学院

刘运

# 主要内容

- 2.1 线性表的定义和特点
- 2.2 案例引入
- 2.3 线性表的类型定义
- 2.4 线性表的顺序表示和实现
- 2.5 线性表的链式表示和实现
- 2.6 顺序表和链表的比较
- 2.7 线性表的应用
- 2.8 案例分析与实现

# 学习要点

- 了解线性表逻辑结构的特征；
- 重点掌握线性表的顺序存储结构和链式存储结构；
- 掌握在顺序存储结构下，线性表的基本操作实现的算法；
- 掌握在链式存储结构下，线性表的基本操作实现的算法；
- 比较线性表两类存储结构的不同特点及适用场合。

# 2.1 线性表的定义和特点

## 一 线性表的定义

线性表是n 个类型相同数据元素的有限序列， 通常记作（a1, a2, a3, …, an ）。

例1、数学中的数列（11，13，15，17，19，21）

例2、英文字母表（A, B, C, D, E… Z ）。

例3、某单位的电话号码簿。

电话号码簿是数据元素的有限序列，每一数据元素包括两个数据项，一个是用户姓名，一个是对应的电话号码。

| 姓名 | 电话号码 |
|---|---|
| 蔡颖 | 63214444 |
| 陈红 | 63217777 |
| 刘建平 | 63216666 |
| 王小林 | 63218888 |
| 张力 | 63215555 |
| … | |

# 说 明

设 A=（$a_1$, $a_2$, … , $a_{i-1}$， $a_i$， $a_{i+1}$, …， $a_n$ ）是一线性表

- 同一线性表中的元素必须是同一类型的；
- 称$a_{i-1}$ 是$a_i$ 的直接前驱，$a_{i+1}$ 是$a_i$ 的直接后继；
- 在线性表中，除第一个元素和最后一个元素之外，其他元素都有且仅有一个直接前驱，有且仅有一个直接后继，具有这种结构特征的数据结构称为线性结构。线性表是一种线性数据结构；
- 线性表中元素的个数n称为线性表的长度，n=0时称为空表；
- $a_i$是线性表的第i 个元素，称i 为数据元素$a_i$ 的位序。

# 2.2 案例引入

## 案例2.1：一元多项式的运算

$$P_n(x) = p_0 + p_1x + p_2x^2 + \ldots + p_nx^n$$

线性表P = ($p_0$,$p_1$,$p_2$,…,$p_n$)

$$P(x) = 10 + 5x - 4x^2 + 3x^3 + 2x^4$$

数组表示

（每一项的指数$i$隐含在其系数$p_i$的序号中）

| 指数(下标i) | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 系数p[i] | **10** | **5** | **-4** | **3** | **2** |

$R_n(x) = P_n(x) + Q_m(x)$


线性表R = ($p_0 + q_0$,$p_1 + q_1$,$p_2 + q_2$,…,$p_m + q_m$,$p_{m+1}$,…,$p_n$)

稀疏多项式

$S(x) = 1 + 3x^{10000} + 2x^{20000}$

# 2.3 线性表的类型定义

**ADT List {**

**数据对象：D={ $a_i$ | $a_i$ ∈ ElemSet, i=1, 2, …， n, n>=0}**

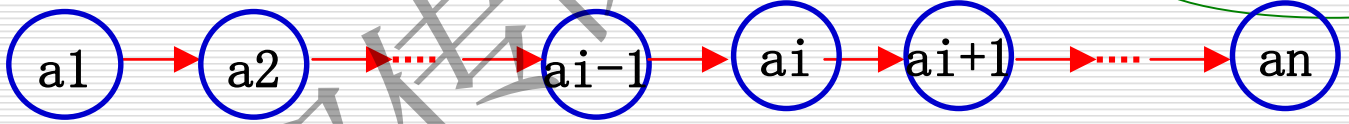
**数据关系：R1={< $a_{i-1}$, $a_i$> | $a_{i-1}$, $a_i$ ∈ D,i=2, …， n}**

前驱关系

**或： R2={< $a_i$, $a_{i+1}$> | $a_i$, $a_{i+1}$ ∈ D, i=1, 2, …， n-1}**

后继关系

图示表示




**顶点：表示数据元素**

**边：表示数据元素间的关系**

# 基本操作：

1 初始化操作 InitList(&L)

操作结果：构造一个空的线性表L。

2 销毁操作DestroyList(&L)

初始条件：线性表L已存在。

操作结果：销毁线性表L。

3 置空操作ClearList(&L)

初始条件：线性表L已存在。

操作结果：将L重置为空表。

4 判空操作ListEmpty(L)

初始条件：线性表L已存在。

操作结果：若L为空表返回TRUE，否则返回FALSE。

5　求表长操作 ListLength(L)

初始条件：线性表L已存在。

操作结果：返回L中元素个数。

6　取元素操作：GetElem(L, i, &e)

初始条件：线性表L已存在，1≤i≤ListLength(L)。

操作结果：用e返回L中第i 个数据元素的值。

7　查找操作 LocateElem (L, e )

初始条件：线性表L已存在。

操作结果：返回L中第1 个与e相同的数据元素的位序。若表中不存在这样的元素，则返回0；

## 8　求前驱操作PriorELem(L, cur_e,&pre_e)

初始条件：线性表L已存在。

操作结果：若cur_e是L的数据元素，且不是第一个，则用pre_e返回它的前驱，否则操作失败，pre_e无定义。

## 9　求后继操作NextELem(L, cur_e,&next_e)

初始条件：线性表L已存在。

操作结果：若cur_e是L的数据元素，且不是最后一个，则用next_e返回它的前驱，否则操作失败，next_e无定义。

## 10 插入操作 ListInsert(&L, i, e )

初始条件：线性表L已存在，1≤i≤ListLength(L)+1。

操作结果：在L中第i个位置之前插入一个新元素e;

## 11 删除操作 ListDelete(&L, i )

初始条件：线性表L已存在且非空，1≤i≤ListLength(L)。

操作结果：删除L的第i个元素，L的长度减1;

## 12 遍历操作 ListTraverse (L )

初始条件：线性表L已存在。

操作结果：对线性表L进行遍历，在遍历过程中对L的每个结点 访问一次。

# 说明

- 上面列出的操作，只是线性表的一些常用的基本操作；
- 不同的应用，基本操作可能是不同的
- 线性表的复杂操作可通过基本操作实现；

这有点类似于数中的情形，例如整数的基本操作是+,-,×，/，如果要求某班同学的平均年龄则可利用+ / 实现，

全班同学的平均年龄=(age1+age2+age3+…)/全班同学的人数

# 2.4 线性表的顺序表示和实现

如何在计算机中存储线性表？

如何在计算机中实现线性表的基本操作？

为了存储线性表，至少要保存两类信息：

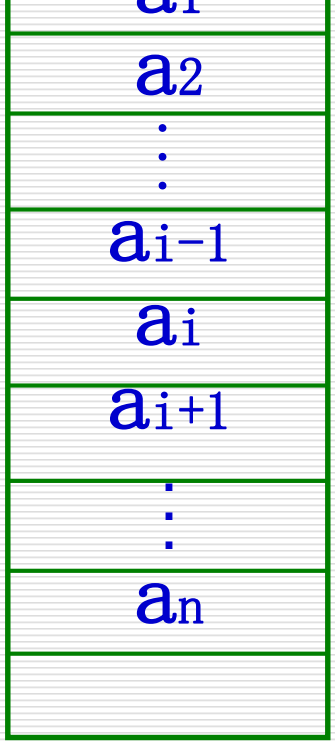
1）线性表中的数据元素；

2）线性表中数据元素的关系；

# 2.4.1线性表的顺序存储表示

线性表的顺序存储结构，就是用一组连续的内存单元依次存放线性表的数据元素。

用顺序存储结构存储的线性表——称为顺序表




用顺序表存储线性表时，数据元素之间的逻辑关系，是通过数据元素的存储顺序反映出来的。

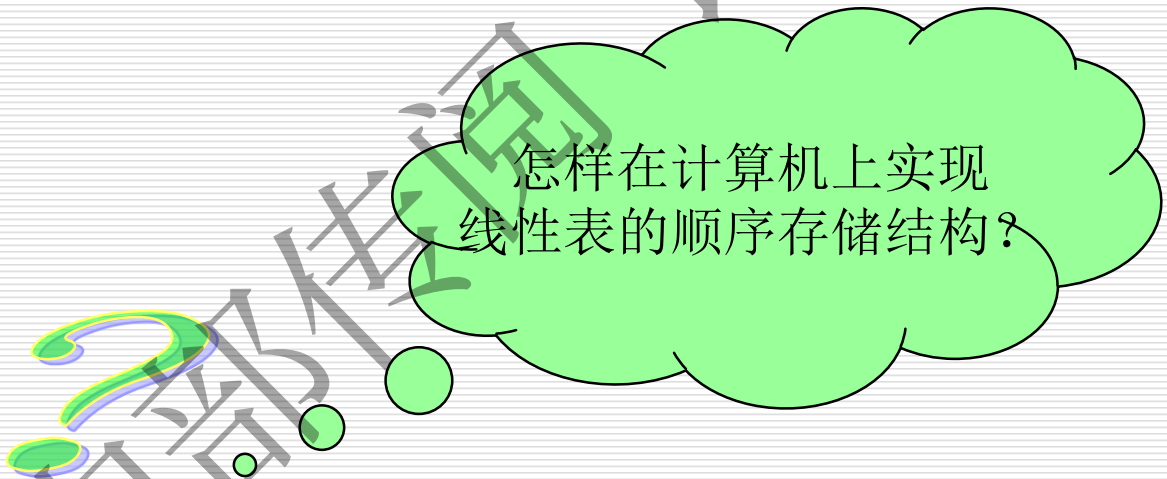
# 说明：

- 在顺序存储结构下，线性表元素之间的逻辑关系，可通过元素的存储顺序反映（表示）出来，所以只需存储数据元素的信息；
- 假设线性表中每个数据元素占用 L 个存储单元，那么，在顺序存储结构中，线性表的第i个元素的存储位置与第1个元素的存储位置的关系是：

$$Loc(a_i) = Loc(a_1) + (i - 1) \times L$$

这里 $Loc(a_i)$是第 i 个元素的存储位置, $Loc(a_1)$ 是第1个元素的存储位置，也称为线性表的基址；

C语言一维数组的机内表示也是顺序结构，因此，可借用C语言的一维数组实现线性表的顺序存储。



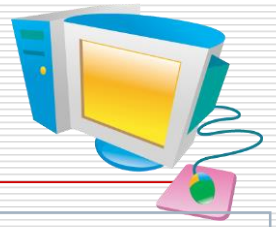
# 顺序表的存储结构

```
#define   LIST_INIT_SIZE   100    //  线性表可能达到的最大长度
typedef struct{
   ElemType   * elem;     //线性表存储空间基址
   int    length;    //当前线性表长度
}SqList;
```

**注：**

***elem：存放线性表元素的一维数组基址；其存储空间在初始化操作（建空表）时动态分配；**

# 复习：



若p已定义为指针变量，且p的初值为&a[0]或a（a为数组名），则

（1）p+i和a+i就是a[i]的地址，或者说它们指向数组的第i+1个元素（下标从0开始）。

（2）*(p+i)或*(a+i)即a[i].

（3）指向数据的指针变量也可以带下标。如p[i]与*(p+i)等价。

由此可知：

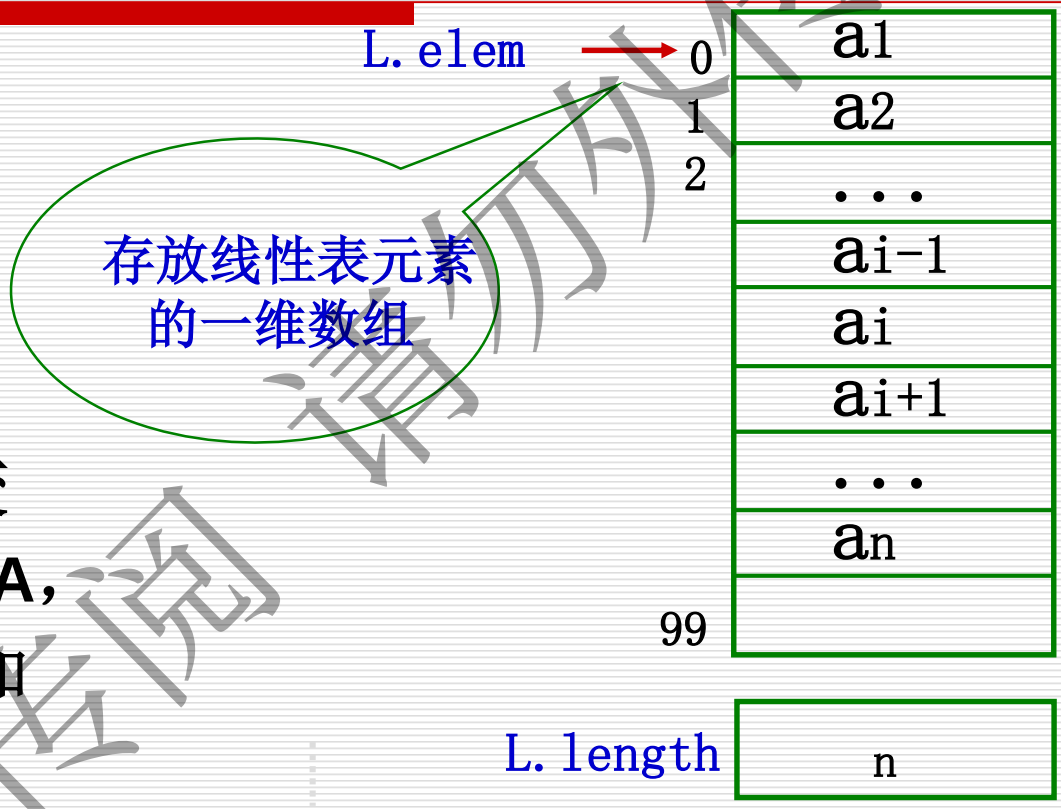
若L是SqList类型的顺序表，则表中第i个数据元素是L.elem[i-1]，即a[i-1]。

（最后一个元素是L.elem[L.Length-1]）

设A =

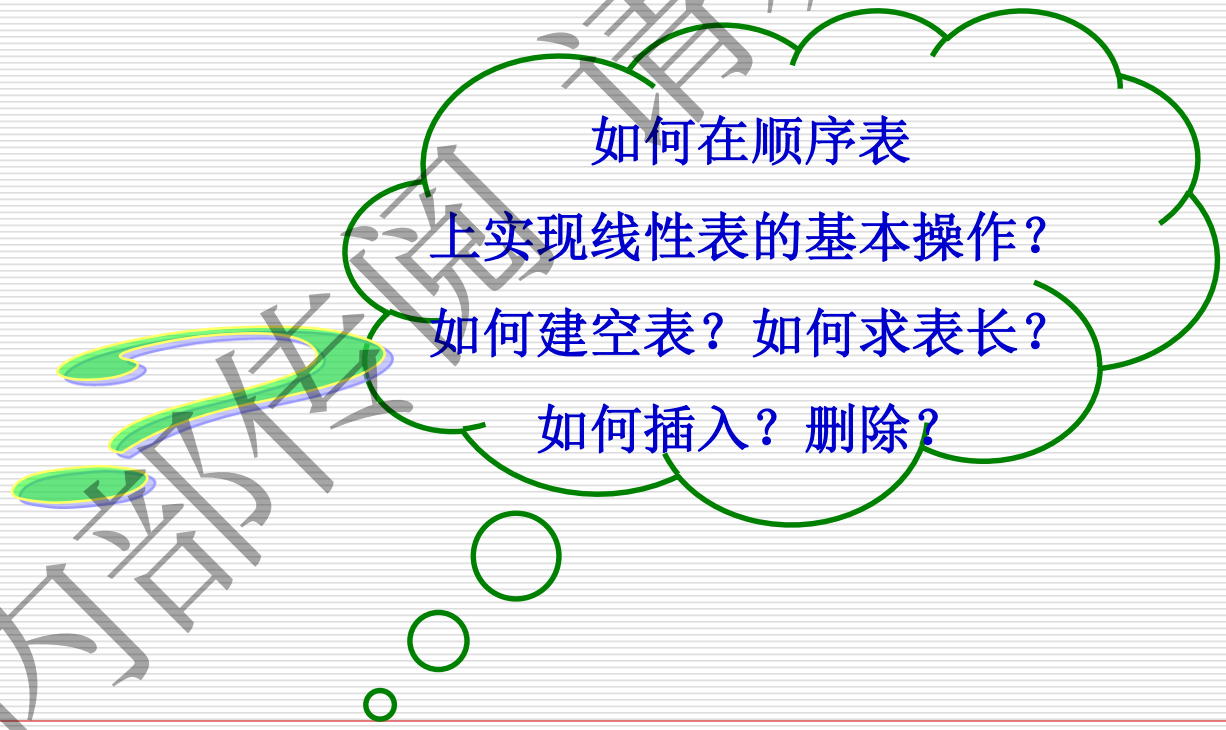
（$a_1$,$a_2$, $a_3$, ... $a_n$ ）

是一线性表，L是SqList 类型的结构变量，用于存放线性表A，则L在内存中的状态如图所示：




顺序表图示

当线性表用顺序表存储时，对线性表各种基本操作实际上就是对存储在内存中的顺序表进行操作。



# 2.4.2 顺序表基本操作的实现

## 1）初始化操作 InitList ( SqList &L)

参数：L是存放线性表的结构变量（称L为顺序表），因为初始化操作对顺序表L进行了修改,所以用了引用参数&L;

功能：建立空的顺序表L

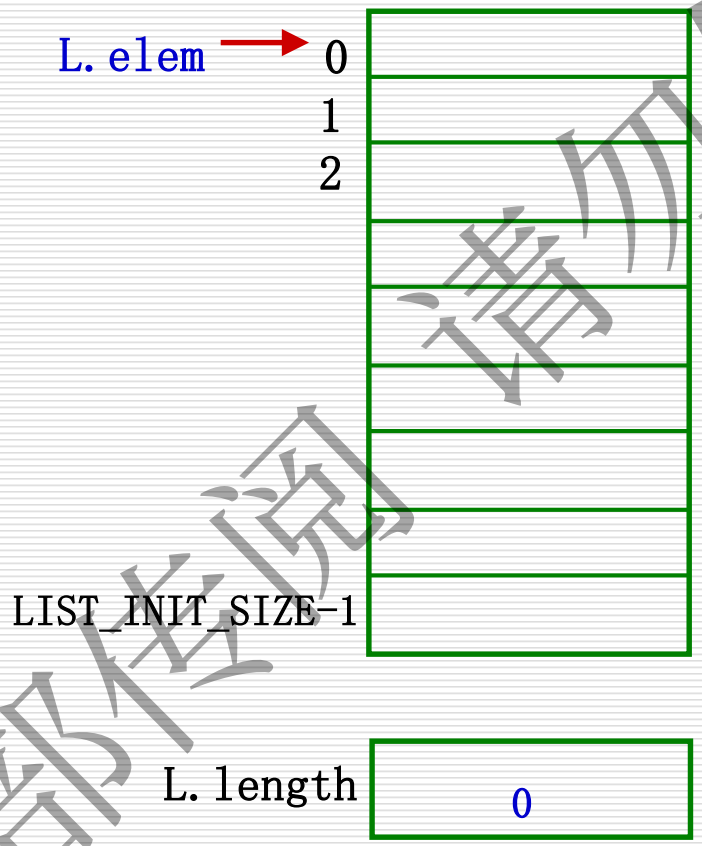
## 算法步骤:

(1) 为顺序表L动态分配一个预定义大小 空间,使L.elem 指向这段空间的基址;

(2) 将表的当前长度设为0。

初始化操作演示

# 初始化操作演示



# 初始化操作算法描述

```
Status InitList (SqList &L){
  //构造一个空的顺序表L
   L.elem=new ElemType[MAXSIZE];
          //为顺序表分配一个大小为MAXSIZE的数组空间
   if (! L.elem) exit(OVERFLOW);  //存储分配失败退出
   L. length=0;  //空表长度为0
   Return OK;
  }
```

算法2.1

# 补充：几个简单基本操作的算法实现

## ■ 销毁线性表L

```
void DestroyList(SqList &L)
{
  if (L.elem) delete[]L.elem;    //释放存储空间
}
```

## ■ 清空线性表L

```
void ClearList(SqList &L)
{
  L.length=0;              //将线性表的长度置为0
}
```

# 补充：几个简单基本操作的算法实现

## ■ 求线性表L的长度

```
int GetLength(SqList L)
{      return (L.length);
}
```

## ■ 判断线性表L是否为空

```
int IsEmpty(SqList L)
{
  if (L.length==0) return 1;
   else return 0;
}
```

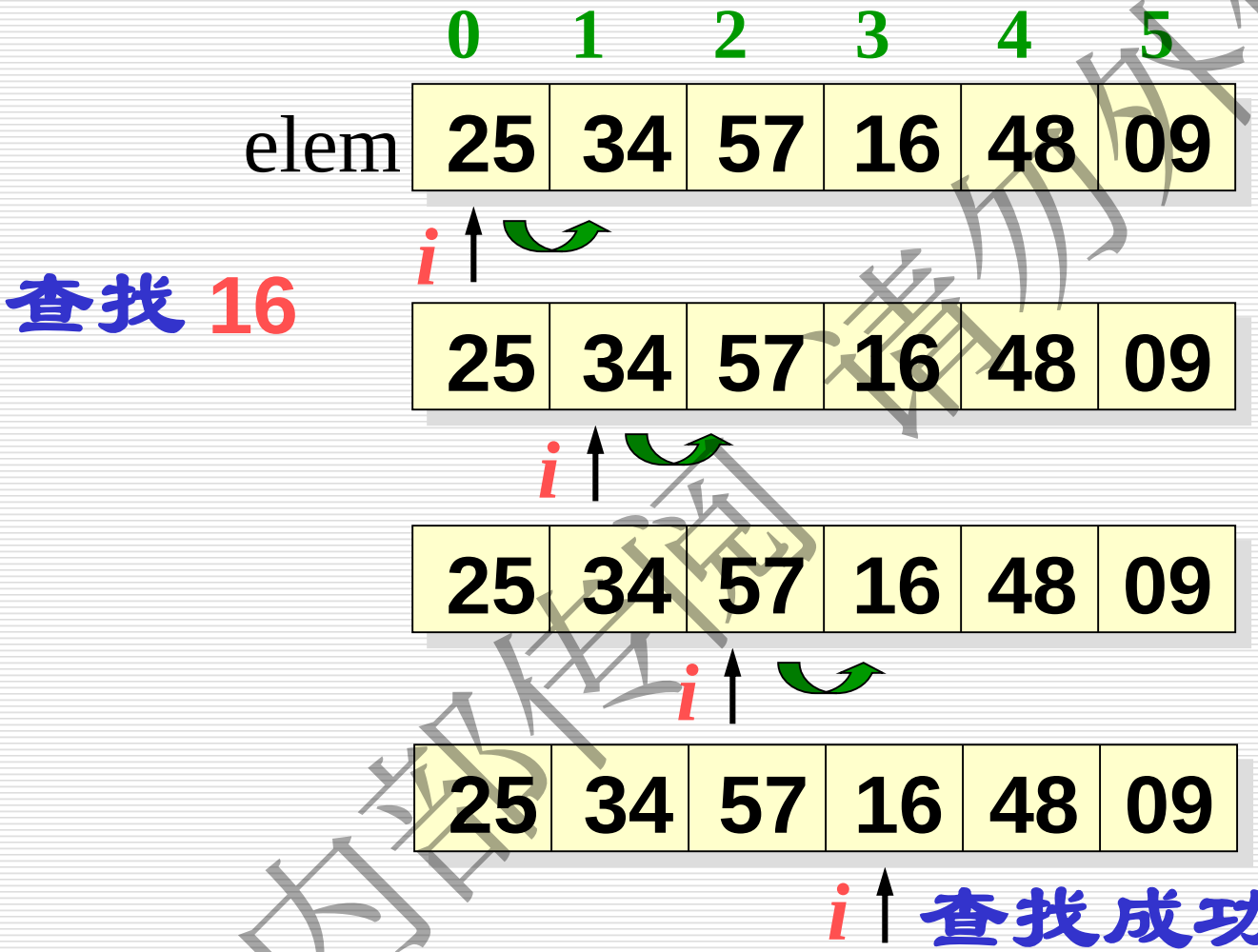
# 2）取值操作算法描述

```
int GetElem(SqList L, int i, ElemType &e)
{//用e返回顺序表L的第i个数据元素的值
    if (i<1||i>L.length) return ERROR;
     //判断i值是否合理，若不合理，返回ERROR
    e=L.elem[i-1];  //第i-1的单元存储着第i个数据
    return OK;
}
```

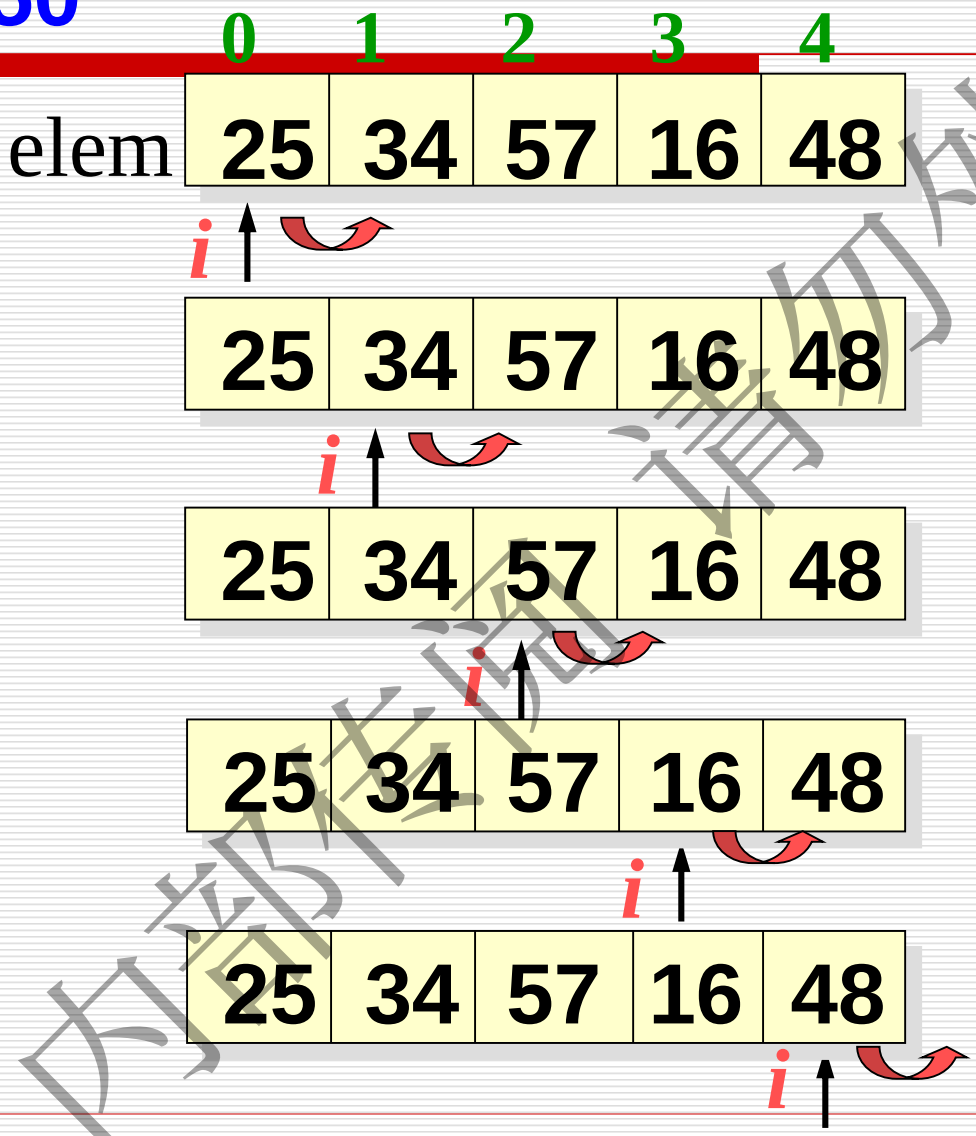
随机存取

算法2.2

# 3）查找操作



# 查找 50



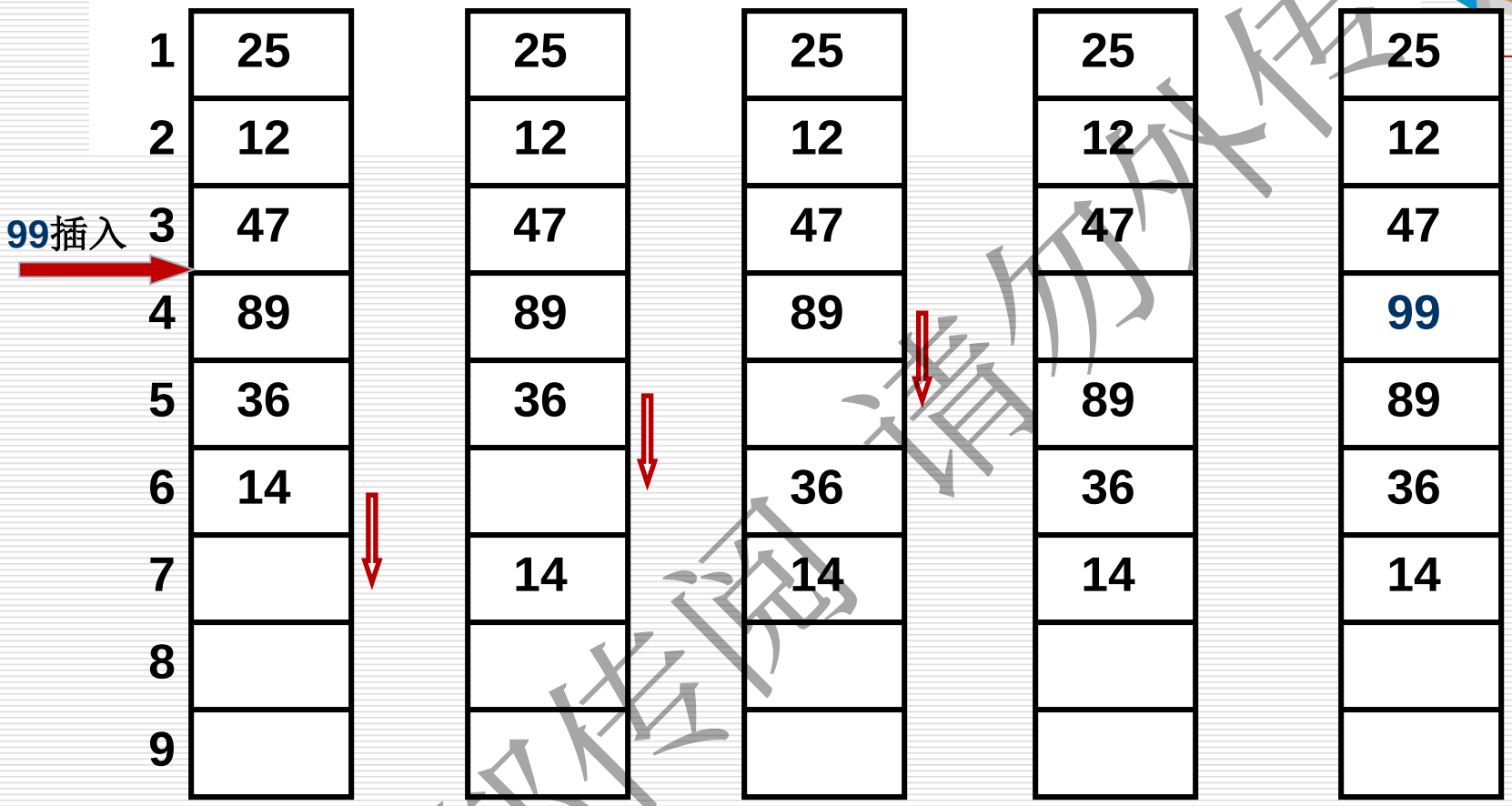

查找失败

# 查找操作算法描述

```
int LocateELem(SqList L,ElemType e)
{//在线性表L中查找值为e的数据元素
    for (i=0;i< L.length;i++)
      if (L.elem[i]==e) return i+1;
    return 0;
}
```

算法2.3

# 4）插入操作（插在第i个元素之前）




**插第 4 个结点之前，移动 6-4+1 次**
**插在第 i 个结点之前，移动 n-i+1 次**

## 插入操作主要步骤：

1）i 是否合法，若合法转2）否则算法结束，返回ERROR；

2）L是否已满，若未满转3）否则算法结束，返回ERROR；

3）将第n至第i（共n-i+1）个元素向后移动一个位置；

4）将新元素写入空出的位置；

5）表长+1 ；

# 插入操作算法描述

```
Status ListInsert (SqList &L,int i ,ElemType e)
{ //在线性表L中第i个数据元素之前插入数据元素e
  if(i<1 || i>L.length+1) return ERROR;        //i值不合法
  if(L.length==MAXSIZE) return ERROR; //存储空间已满
  for(j=L.length-1;j>=i-1;j--)
     L.elem[j+1]=L.elem[j];    //插入位置及之后的元素后移
  L.elem[i-1]=e;                //将新元素e放入第i个位置
   ++L.length;                          //表长增1
  return OK;
}
```

算法2.4

# 算法分析

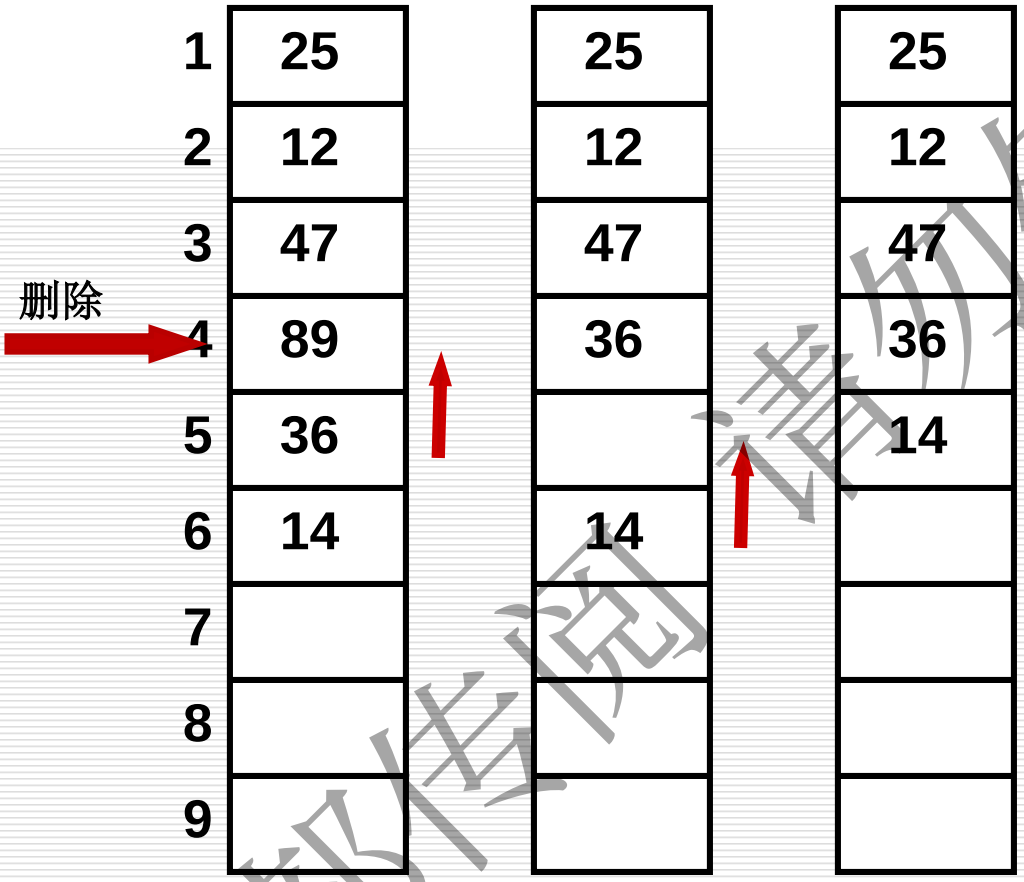
## 算法时间主要耗费在移动元素的操作上

- 若插入在尾结点之后，则根本无需移动（特别快）；
- 若插入在首结点之前，则表中元素全部后移（特别慢）；
- 若要考虑在各种位置插入（共n+1种可能）的平均移动次数，该如何计算？

$$E_{ins} = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{1}{n+1}(n+\cdots+1+0)$$

$$= \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

# 5）删除操作（删除第i个元素）




**删除第 4 个结点，移动 6-4 次**
**删除第 i 个结点，移动 n-i 次**

## 删除操作主要步骤：

(1) 判断删除位置i 是否合法（合法值为1≤i≤n）。

(2) 将欲删除的元素保留在e中。

(3) 将第i+1至第n 位的元素依次向前移动一个位置。

(4) 表长减1，删除成功返回OK。

# 算法描述

```
Status ListDelete (SqList &L,int i)
{//将线性表L中第i个数据元素删除
  if((i<1)||(i>L.length)) return ERROR;     //i值不合法
  for (j=i; j<=L.length-1; j++)
       L.elem[j-1]=L.elem[j];       //被删除元素之后的元素前移
  --L.length;                       //表长减1
 return OK;
}
```

算法2.5

# 算法分析

## 算法时间主要耗费在移动元素的操作上

- 若删除尾结点，则根本无需移动（特别快）；
- 若删除首结点，则表中n-1个元素全部前移（特别慢）；
- 若要考虑在各种位置删除（共n种可能）的平均移动次数，该如何计算?

$$E_{del} = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{1}{n}\frac{(n-1)n}{2} = \frac{n-1}{2}$$

# 小结

**顺序表的特点：**

- 顺序存储；
- 随机存取；
- 插入删除操作要通过移动元素实现；

# 作业

问题：两个非递减排列的顺序表合并为一个新表

## 合并操作主要步骤：

1）为新表分配空间，如果分配失败算法结束；

2）合并操作，通过比较两个顺序表中当前数据元素的值；

3）插入顺序表中未归并的元素；

4）修改新表的表长

# 合并操作算法描述

```
void MergeList_Sq(SqList La, SqList Lb, SqList &Lc)
{//归并两个非递减排列的顺序表，归并后的元素也按非递减排列
    pa=La.elem; pb=Lb.elem;
    Lc.listsize=La.length+Lb.length;
    //分配空间
    Lc.elem=(ElemType*)malloc(Lc.listsize*sizeof(ElemType));
    if(!Lc.elem) exit(OVERFLOW);
    pc=Lc.elem;
    pa_last = La.elem+La.length-1;
    pb_last=Lb.elem+Lb.lenght-1;
```

# 合并操作算法描述

```
void MergeList_Sq(SqList La, SqList Lb, SqList &Lc)
{//归并两个非递减排列的顺序表，归并后的元素也按非递减排列
    //归并操作
    while(pa<pa_last && pb<pb_last){
    if(*pa<*pb) *pc++ = *pa++;
    else *pc++=*pb++;}
    //插入剩余元素
    while(pa<pa_last) *pc++=*pa++;
    while(pb<pb_last) *pc++=*pb++;
    Lc.length = La.length + Lb.length;//修改归并后的表长
}
```

# 2.5 线性表的链式存储和实现

线性表的链式存储结构是用一组任意的存储单元存储线性表的各个数据元素。为了表示线性表中元素的先后关系，每个元素除了需要存储自身的信息外还需保存直接前驱元素或直接后继元素的存储位置。

# 2.5.1 线性链表

## 一　线性链表的概念

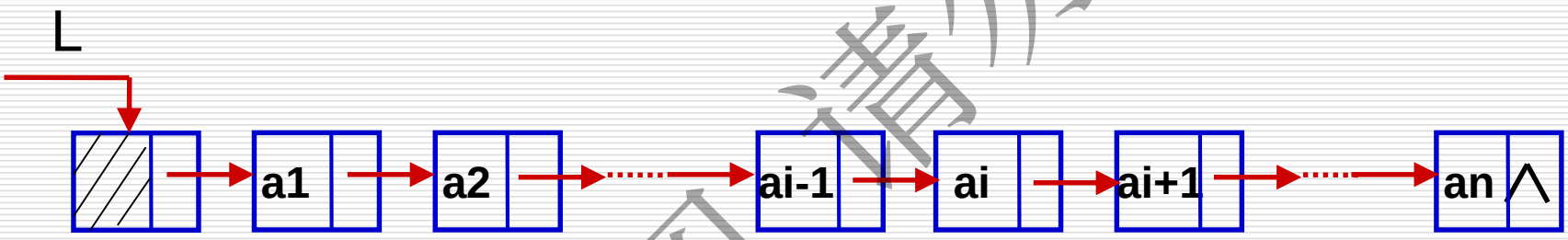
用一组任意的存储单元存储线性表中的数据元素，对每个数据元素除了保存自身信息外，还保存了直接后继元素的存储位置。

| | | |
|---|---|---|
| 1010 | a4 | 0 |
| 1012 | | |
| 1014 | a3 | 1010 |
| 1016 | | |
| 1018 | | |
| 1020 | a1 | 1024 |
| 1022 | | |
| 1024 | a2 | 1014 |
| 1026 | | |

用线性链表存储线性表时，数据元素之间的关系是通过保存直接后继元素的存储位置来表示的。

# 线性链表图示

为直观起见，通常用如下所示的图表示链表，其中，箭头表示相应单元中保存的是它所指向结点的存储地址。




线性链表图示

用线性链表存储线性表时，数据元素之间的关系是通过保存直接后继元素的存储位置来表示的

# 线性链表有关术语

结点：数据元素及直接后继的存储位置（地址）组成一个数据元素的存储结构，称为一个结点；

结点的数据域：结点中用于保存数据元素的部分；

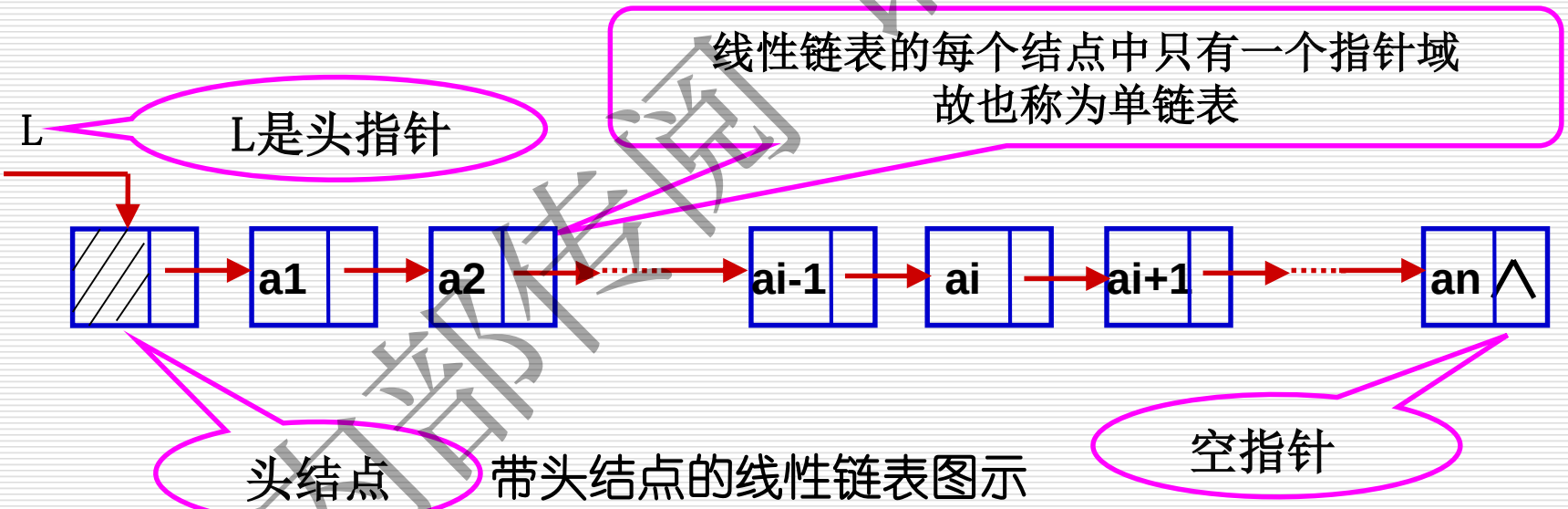
结点的指针域：结点中用于保存数据元素直接后继存储地址的部分；

头指针：用于存放线性链表中第一个结点的存储地址；

空指针：不指向任何结点，线性链表最后一个结点的指针通常是空指针；

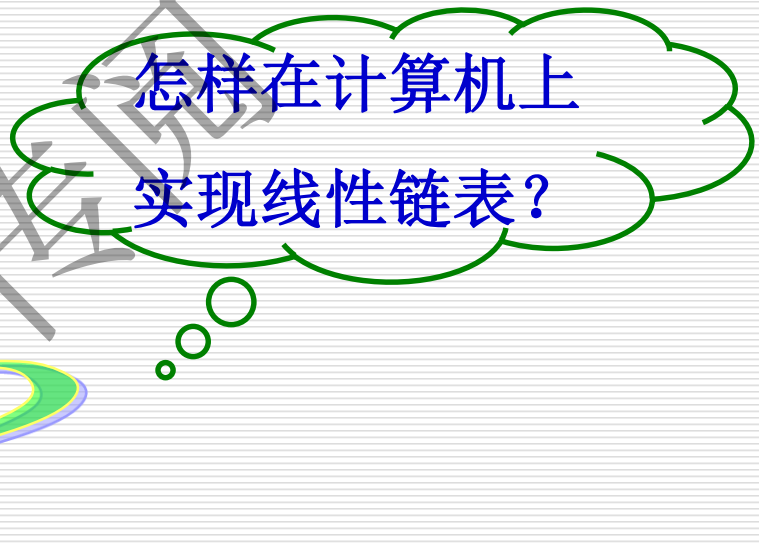
头结点：线性链表的第一元素结点前面的一个附加结点，称为头结点；

带头结点的线性链表：第一元素结点前面增加一个附加结点的线性链表称为 带头结点的线性链表；




带头结点的线性链表图示

可以用C语言的结构体表示线性链表的结点，用指针存放直接后继的存储地址。

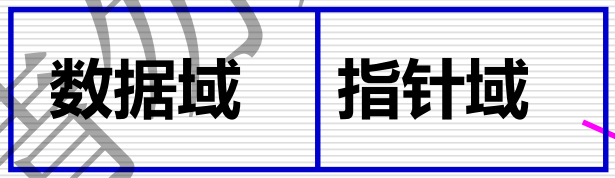


# 线性链表的结点类型定义及指向结点的指针类型定义

```
typedef struct  LNode{
    ElemType   data;
    Struct  LNode  *next;
}LNode, *LinkList;
```

LNode类型结构变量

data next




结点变量图示

LinkList类型指针变量L

**注：**

LNode：结构类型名；

LNode类型结构变量有两个域：

data：用于存放线性表的数据元素，

next：用于存放元素直接后继结点的地址；

该类型结构变量用于表示线性链表中的一个结点；

LinkList：指针类型名；

LinkList类型指针变量用于存放LNode类型结构变量的地址；

# 补充说明:

设p是指向结构体变量的指针(如Linklist类型指针),则以下两种形式等价:

(1)(*p).成员名

(2)p->成员名

如: (*p).data ⇔ p->data

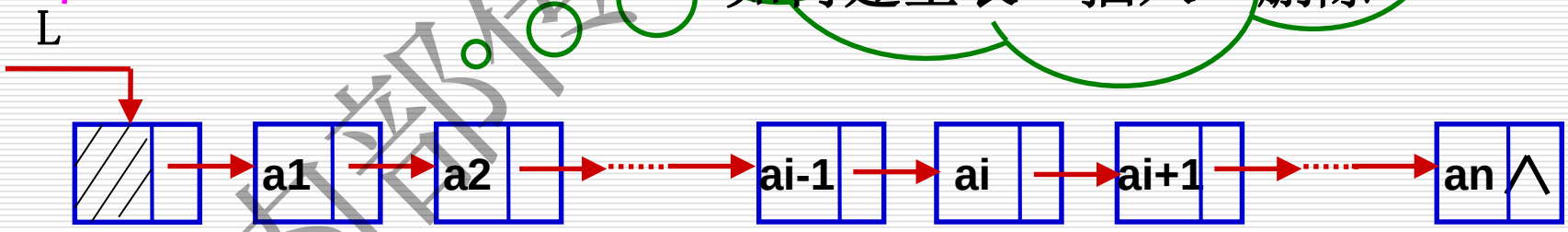
# 2.5.2 单链表基本操作的实现

设L是LinkList类型变量，L用来保存线性链表中第一个结点的地址， 则L为线性链表的头指针。

以L为头指针的线性链表称为线性链表L

如何在线性链表L上实现线性表的基本操作？

如何建空表？插入？删除？



# 1）初始化操作

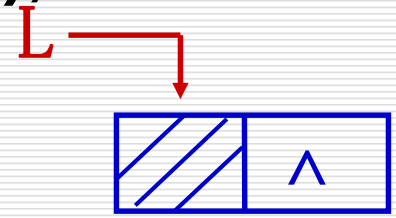
功能： 建空线性链表L

参数： L为线性链表的头指针

主要步骤：调用malloc ( )分配一结点的空间,并将其地址赋值给L

算法：

```
Status InitList_L (LinkList &L) {
L = (LinkList)malloc(sizeof(Lnode));
 If (!L) exit(OVERFLOW);
L->next = null;
Return OK;
}// InitList_L
```



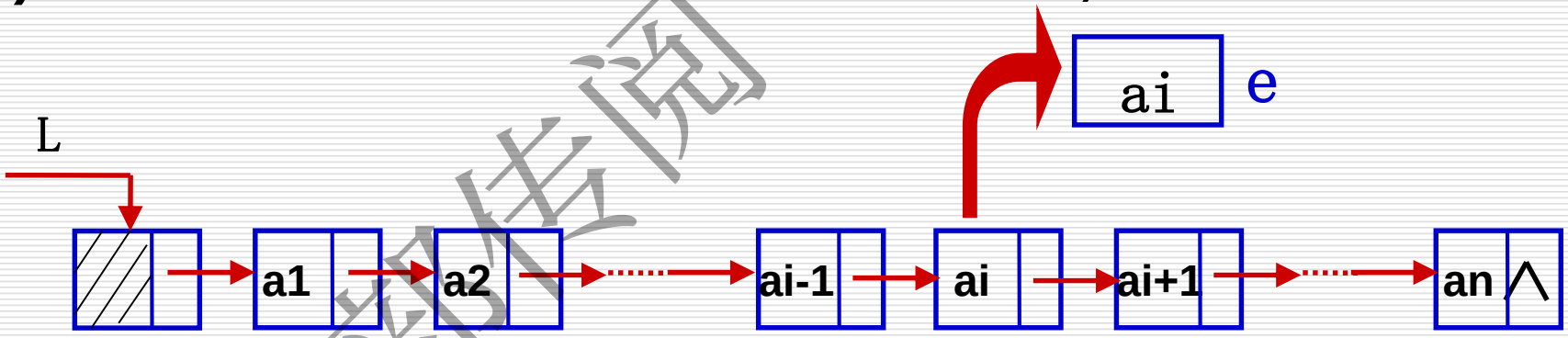
# 2）取元素操作

功能：将线性链表中第i 个元素赋值给 e

主要步骤：

1）查找链表的第 i个元素结点；

2) 将第i个元素结点中的数据元素赋值给e;




取元素元素操作图示

# 取元素操作算法：

```
Status GetElem_L(LinkList L, int I, ElemType &e){
    //L为带头结点的单链表的头指针。
    //当第i个元素存在时，其值赋给e并返回OK，否则返回ERROR
    p=L->next; j=1;  //初始化，p指向第一个结点，j为计数器
    while(p&& j<i){
        //顺指针向后查找，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
```

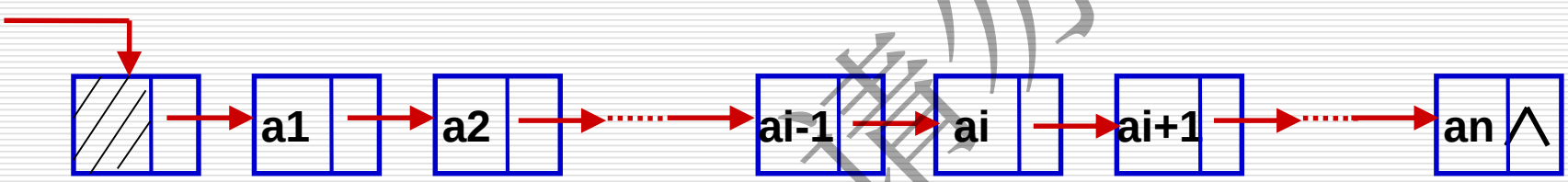


算法 2.8

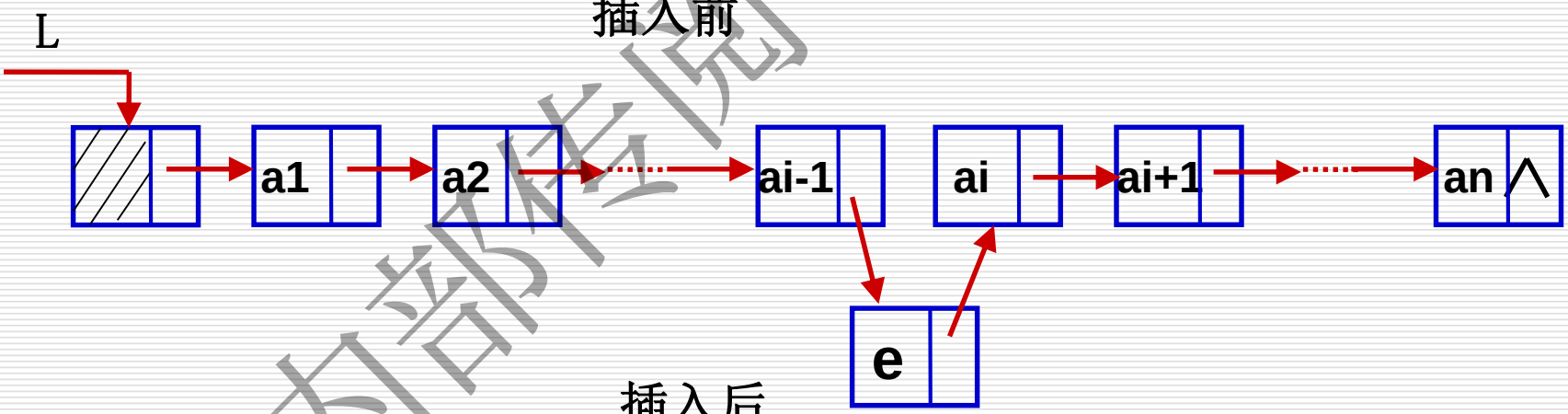
# 3）插入操作

**功能：**

在线性链表L的第i个元素结点之前插入一个新元素结点；




插入前




插入后

# 插入操作主要步骤：

1）查找链表L的第 i-1个元素结点；

2）为新元素建立结点；

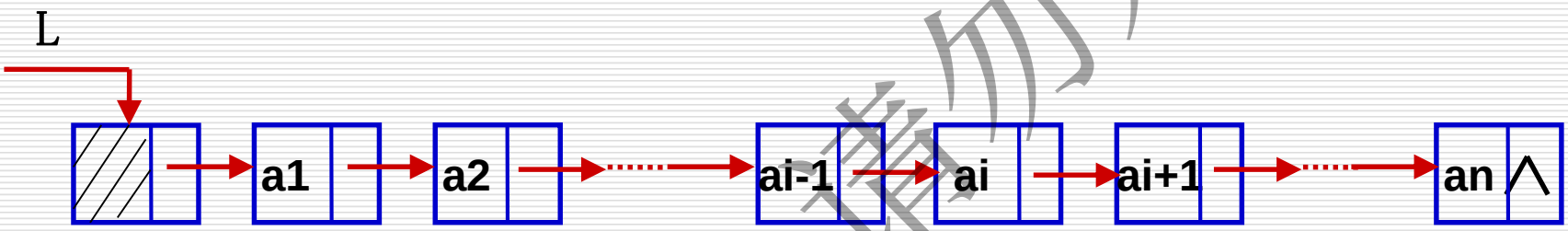
3）修改第 i-1个元素结点的指针和新元素结点指针，完成插入；

# 插入操作算法：

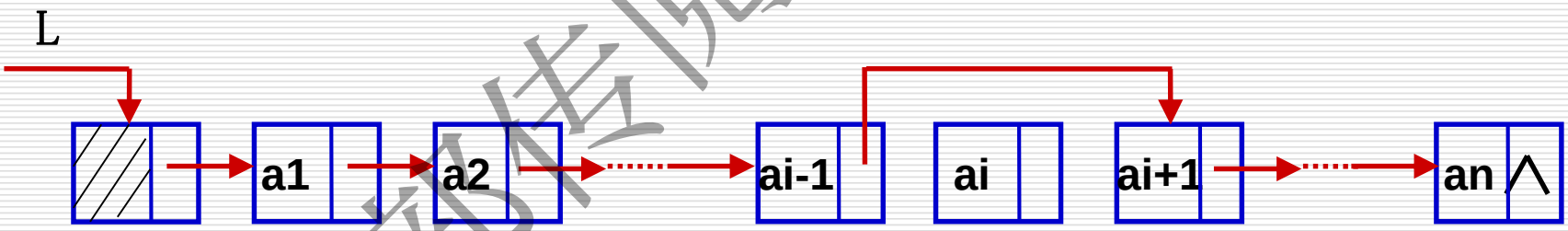
```
Status ListInsert_L(LinkList &L, int i, ElemType e){
  //在带头结点的线性链表L中第I元素结点之前插入元素e
    p=L; j=0
    while (p&&j<i-1){p=p->next; ++j;}//寻找第i-1个元素结点
    if(!p||j>i)return ERROR; // i小于j或者大于表长
    s=(LinkList)malloc(sizeof(LNode));// 分配新结点
     s->data=e;
     s->next=p->next; p->next=s;   //插入新结点
     return OK;
}//LinstInsert_L
```

# 4）删除操作

功能：在线性链表L中删除第i个元素，并且用e 返回其值




删除前




删除后

# 删除操作主要步骤：

1）查找链表的第 i-1个元素结点；

2）修改第 i-1个元素结点指针，删除第i个元素结点；

3) 将第i个元素结点中的数据元素赋值给e;

4）回收被删除结点空间；

# 删除操作算法：

```
Status ListDelete_L(LinkList &L, int i, ElemType &e){
    //在带头结点的单链线性表L中，删除第i个元素，并由e返回其值
    p=L; j=0;
    while (p->next&&j<i-1){
      //寻找第i个结点，并令p指向其前趋
        p=p->next; ++j;     }
    if(!p->next)||j>i-1)return ERROR; //  表中无第i个结点(i不合法)
    q=p->next;p->next=q->next;   //删除结点
    e =q->data; free(q);              //  回收（释放）结点空间
    return OK;
}//LinstDelete_L
```

算法： 2.10

# 头插法建立单链表步骤

1.先建一个空表(带头结点):

```
L=(LinkList)malloc(sizeof(LNode));
L->next=NULL;
```

2.生成新结点:

```
p=(LinkList)malloc(sizeof(LNode));
```

3. 向结点中填入元素值:

```
scanf(&p->data);
```

4.将新结点链入表首(头结点后)

```
p->next=L->next;
```

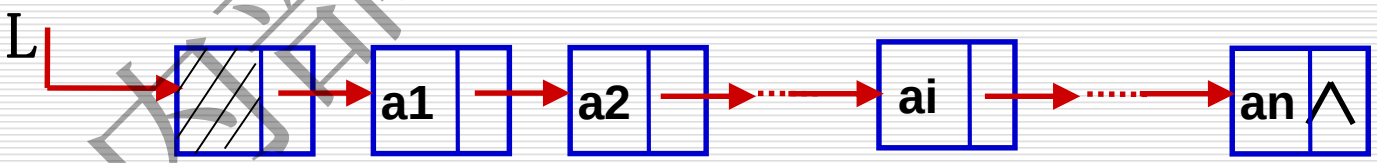
5.改变头结点的指针域.

```
L->next=p;
```

6.重复2到5,直到生成n个元素结点.

# 头插法建立单链表算法

```
void CreateList_L(LinkList &L, int n) {
    //逆序输入n个元素的值，建立带表头结点的线性链表
    L=（LinkList）malloc(sizeof (LNode));
    L->next=NULL; //先建立一个带头结点的单链表
    for (i=n; i>0;--i){
        p=(LinkList)malloc(sizeof(LNode));//生成新结点
        scanf(&p->data);//输入元素值
        p->next=L->next;L->next=p; //插入到表头/  }
}//CreateList_L
```



# 注:

- 此算法是从表尾到表头逆向建立单链表，注意循环语句FOR中i的变化范围是n到1，即第一个插入的应为an。（设线性表为(a1, a2, …an)）。
- 算法的时间复杂度为O(n)。
- 所建的单链表带头结点，由此也可以看出头结点的作用。（使在空表中插入一个元素与在非空表中插入一个元素一致，头指针始终不为空）。

# 作业

## 习题三

1. P13

   2.1，2.2，2.3，2.11,2.14，2.17，2.22

2. 尾插法建立单链表。

# 小结

线性链表的特点

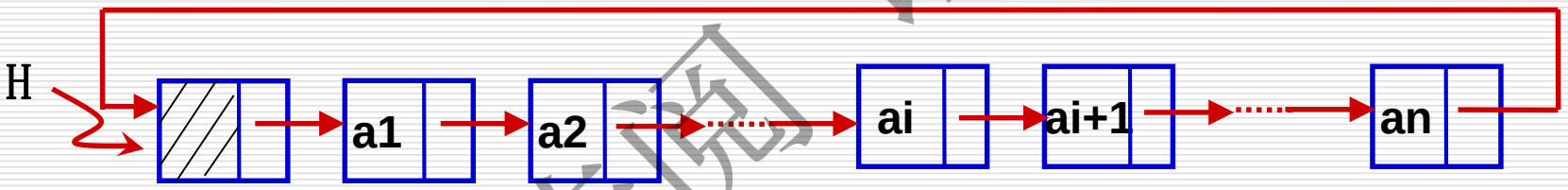
- 链式存储；
- 顺序存取；
- 插入删除操作通过修改结点的指针实现。

# 2.5.3 循环链表

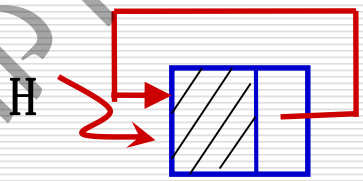
## 1 循环链表的概念

循环链表是线性表的另一种链式存储结构，它的特点是将线性链表的最后一个结点的指针指向链表的第一个结点。

## 2 循环链表图示




(a)非空表




（b）空表

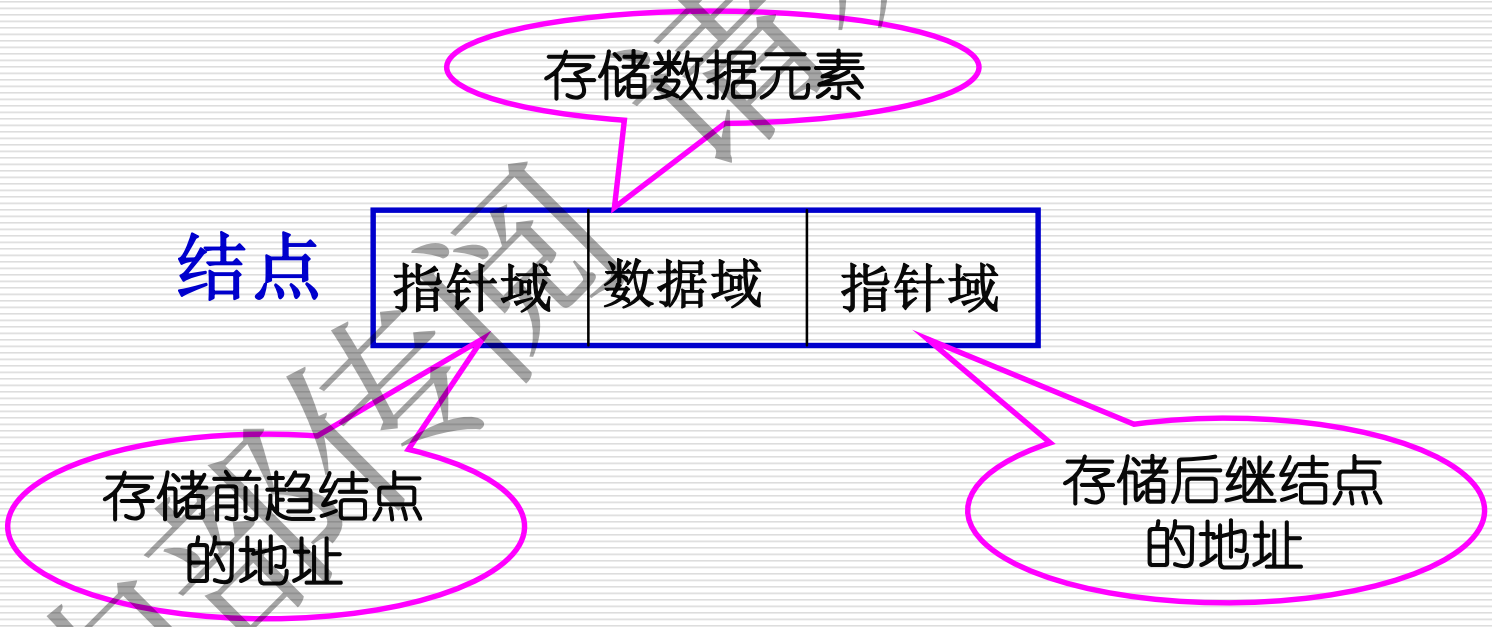
# 说明

- 在解决某些实际问题时循环链表可能要比线性链表方便些。如将一个链表链在另一个链表的后面；
- 循环链表与线性链表操作的主要差别是算法中循环结束的条件；
- 对循环链表，有时不给出头指针，而是给出尾指针，设立尾指针可以很方便的找到线性表的第一个元素和最后一个元素结点，可使某些操作易于实现；例如将两个链表首尾相连的操作；

# 2.5.4 双向链表

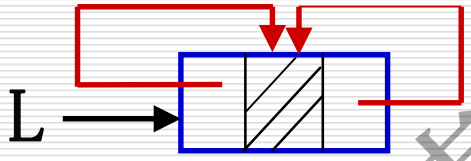
## 1 双向链表的概念

双向链表中，每个结点有两个指针域，一个指向直接后继元素结点，另一个指向直接前趋元素结点。

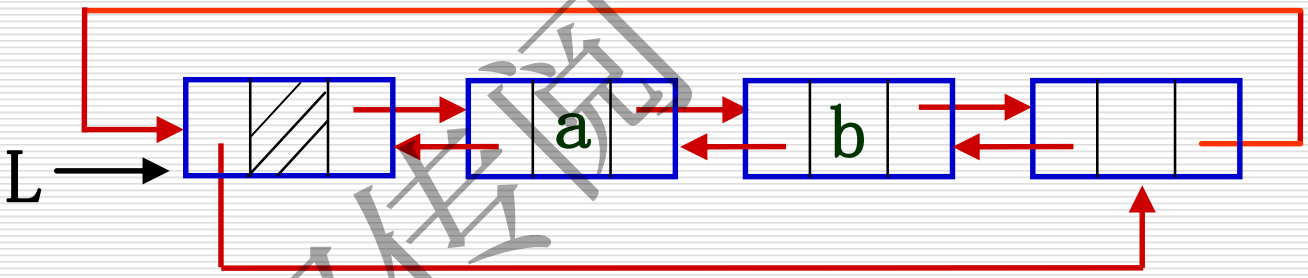



(a) 结点图示

# 双向链表图示

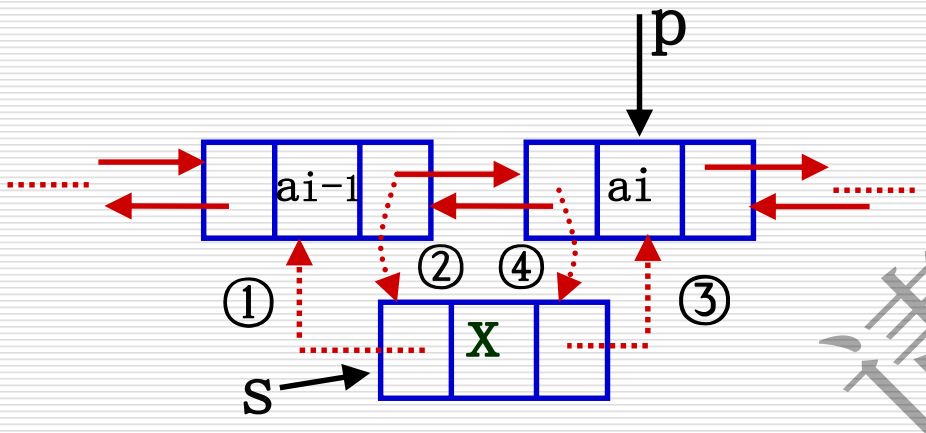



(b)空的双向循环链表

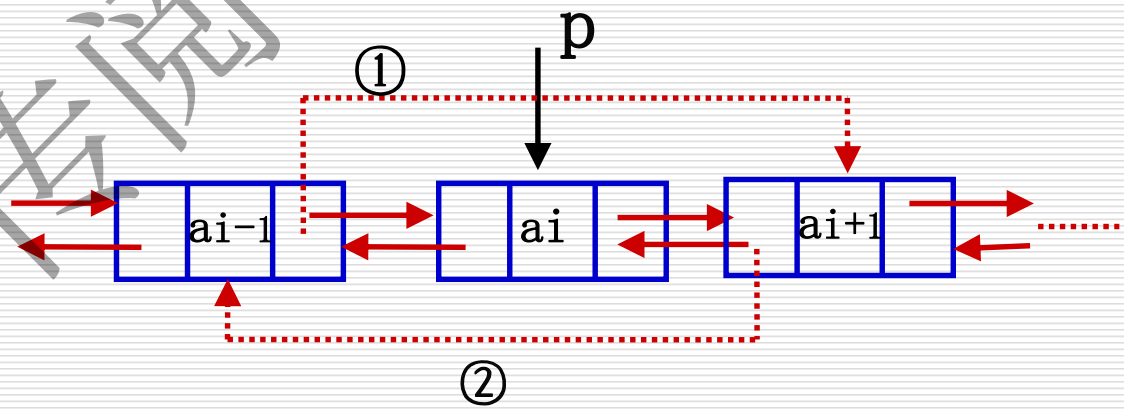



(c) 非空的双向循环链表

# 双向链表的基本操作算法




在双向链表中插入一个结点时指针的变化情况

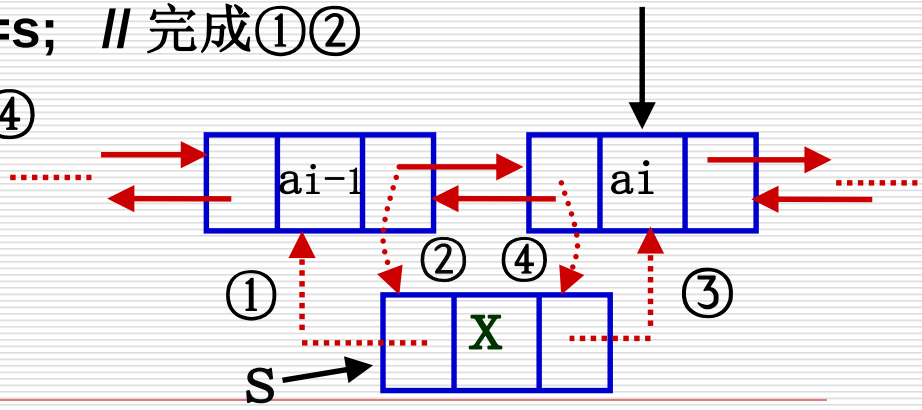



在双向链表中删除结点时指针变化情况

# 1）插入操作算法

```
Status ListInsert_DuL(DuLinkList &L, int i, ElemType e){
//在带头结点的双链循环线性表L中第i个位置之前插入元素e，
//i的合法值为1≤i≤表长+1。
if(!p=GetElemP_DuL(L,i))) //在L中确定第i个元素的位置指针p
return ERROR; //p=NULL，即第i个元素不存在
//建新结点
 if (!(s=(DuLinkList)malloc(sizeof(DuLNode))))return ERROR;
s->data=e;
s->prior=p->prior; p->prior->next=s;  // 完成①②
s->next=p; p->prior=s;   //完成③④
return OK;
}//LinstInsert_DuL
```

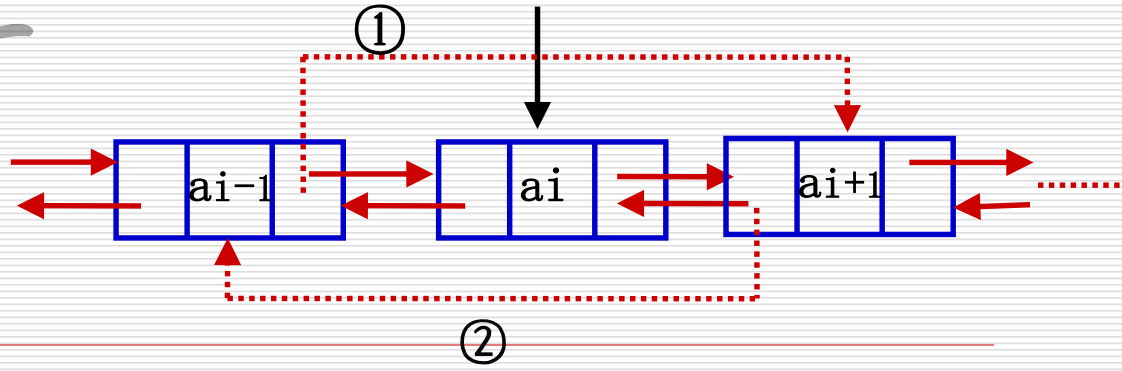
算法2.18



# 2）删除操作算法

```
Status ListDelete_DuL(DuLinkList &L, int i,
ElemType&e){
//删除带头结点的双链循环线性表L的第i个元素，i的合法值为1≤i≤表长
if(!p=GetElemP_DuL(L,i))) //在L中确定第i个元素的位置指针p
return ERROR; //p=NULL，第i个元素不存在
e = p->data;
p->prior->next=p->next;  //完成删除图示中的①
p->next->prior=p->prior;  //完成删除图示中的②
free(p); return OK;
}//LinstDelete_DuL
```

算法2.19



# 补充：静态链表

## 1 静态链表的概念

用一维数组表示的线性链表，称为静态链表。

## 2 静态链表的类型定义

```
#define  MAXSIZE   1000   // 链表的最大长度
typedef struct{
   ElemType   data;
   int              cur;
} component, SLinkList[MAXSIZE];
```

**SLinkList**：数组的类型名；

**SLinkList**类型的数组变量是结构数组，每一数组分量包括两个域：

**data**：用于存储线性表元素；

**cur**：用于存储直接后继元素在数组中的位置（下标）；

# 静态链表图示

| | | |
|---|---|---|
| 0 | | 7 |
| 1 | | 5 |
| 2 | $a_4$ | 0 |
| 3 | | 5 |
| 4 | $a_3$ | 2 |
| 5 | | 6 |
| 6 | | 8 |
| 7 | $a_1$ | 9 |
| 8 | | 10 |
| 9 | $a_2$ | 4 |
| 10 | | 0 |

数组下标

静态链表图示

| | | |
|---|---|---|
| 1010 | $a_4$ | 0 |
| 1012 | | |
| 1014 | $a_3$ | 1010 |
| 1016 | | |
| 1018 | | |
| 1020 | $a_1$ | 1014 |
| 1022 | | |
| 1024 | $a_2$ | 1020 |
| 1026 | | |

地址

线性链表图示

静态链表与线性链表的区别？

# 静态链表操作图示

## 1) 静态链表的插入

插入SHI

| | | |
|---|---|---|
| 0 | | 1 |
| 1 | ZHAO | 2 |
| 2 | QIAN | 3 |
| 3 | SUN | 4 |
| 4 | LI | 5 |
| 5 | ZHOU | 6 |
| 6 | WU | 7 |
| 7 | ZHENG | 8 |
| 8 | WANG | 0 |
| 9 | | |
| 10 | | |

插入前

| | | |
|---|---|---|
| 0 | | 1 |
| 1 | ZHAO | 2 |
| 2 | QIAN | 3 |
| 3 | SUN | 4 |
| 4 | LI | 9 |
| 5 | ZHOU | 6 |
| 6 | WU | 7 |
| 7 | ZHENG | 8 |
| 8 | WANG | 0 |
| 9 | SHI | 5 |
| 10 | | |

插入后

## 2） 静态链表的删除


| | | |
|---|---|---|
| 0 | | 1 |
| 1 | ZHAO | 2 |
| 2 | QIAN | 3 |
| 3 | SUN | 4 |
| 4 | LI | 5 |
| 5 | ZHOU | 6 |
| 6 | WU | 7 |
| 7 | ZHENG | 8 |
| 8 | WANG | 0 |
| 9 | | |
| 10 | | |

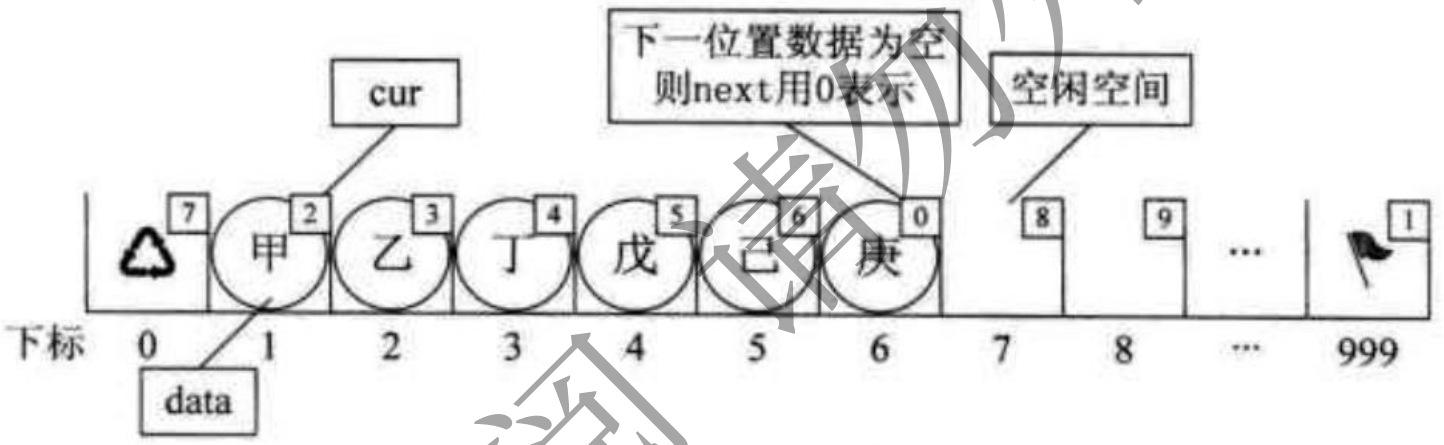
删除前

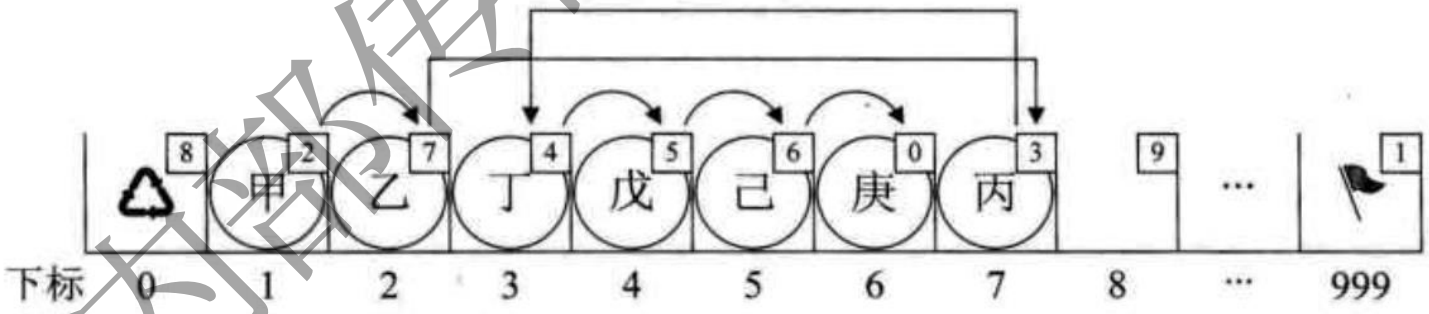
| | | |
|---|---|---|
| 0 | | 1 |
| 1 | ZHAO | 2 |
| 2 | QIAN | 4 |
| 3 | SUN | 4 |
| 4 | LI | 5 |
| 5 | ZHOU | 6 |
| 6 | WU | 7 |
| 7 | ZHENG | 8 |
| 8 | WANG | 0 |
| 9 | | |
| 10 | | |

删除后

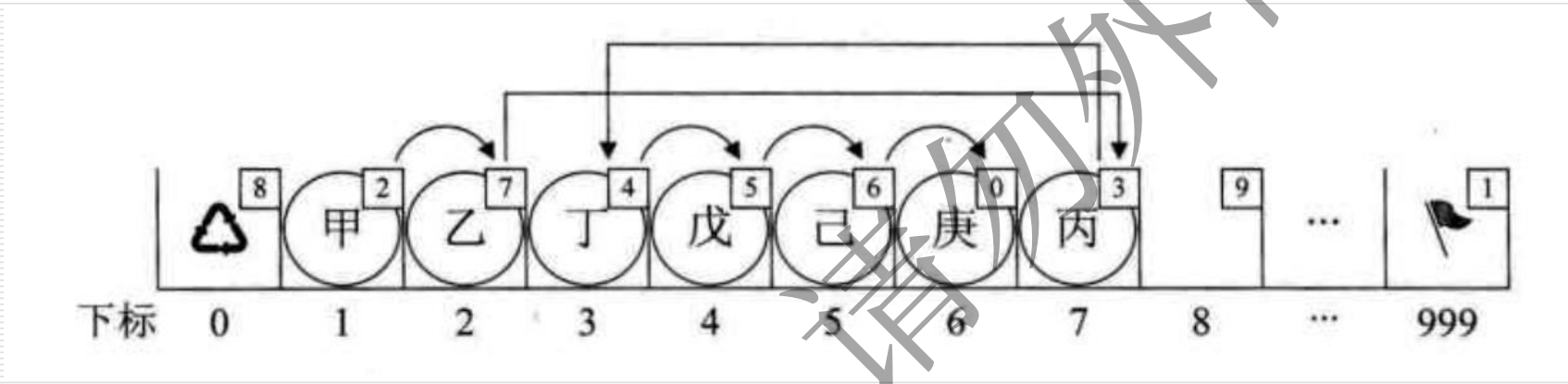
# 静态链表插入元素

## 1 静态链表插入元素

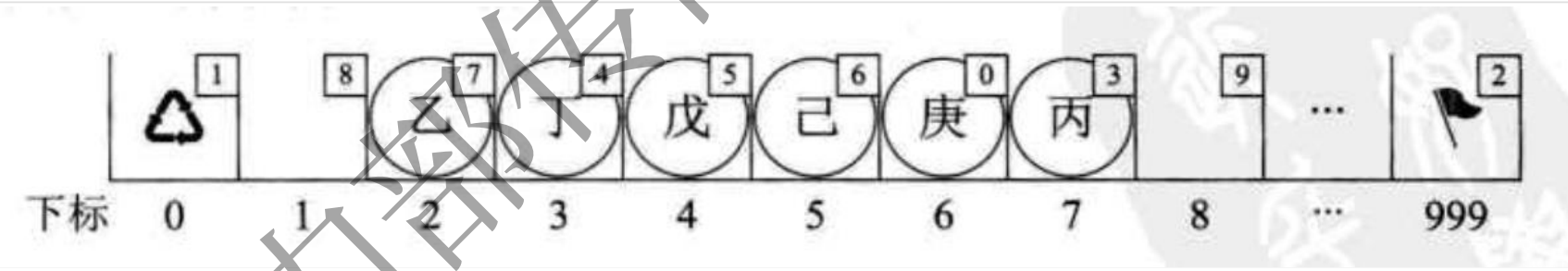


# 静态链表删除元素




删除甲



# 2.6 顺序表和链表的比较

| 存储结构<br>比较项目 | | 顺序表 | 链表 |
|---|---|---|---|
| 空间 | 存储空间 | 预先分配，会导致空间闲置或溢出现象 | 动态分配，不会出现存储空间闲置或溢出现象 |
| | 存储密度 | 不用为表示结点间的逻辑关系而增加额外的存储开销，存储密度等于1 | 需要借助指针来体现元素间的逻辑关系，存储密度小于1 |
| 时间 | 存取元素 | 随机存取，按位置访问元素的时间复杂度为O(1) | 顺序存取，按位置访问元素时间复杂度为O($n$) |
| | 插入、删除 | 平均移动约表中一半元素，时间复杂度为O($n$) | 不需移动元素，确定插入、删除位置后，时间复杂度为O(1) |
| 适用情况 | | ① 表长变化不大，且能事先确定变化的范围<br>② 很少进行插入或删除操作，经常按元素位置序号访问数据元素 | ① 长度变化较大<br>② 频繁进行插入或删除操作 |

# 2.7 线性表的应用

## 将两个有序线性链表归并成一个有序表

设线性表A、B分别用头指针为La 、 Lb 的两个带头结点的线性链表存储，且两线性链表中元素按非递减顺序排列，编写算法，将La 、 Lb归并得到线性链表Lc，Lc中的元素也按值非递减顺序排列。（注意：线性链表Lc中的结点利用原La，Lb的结点）

基本思想：

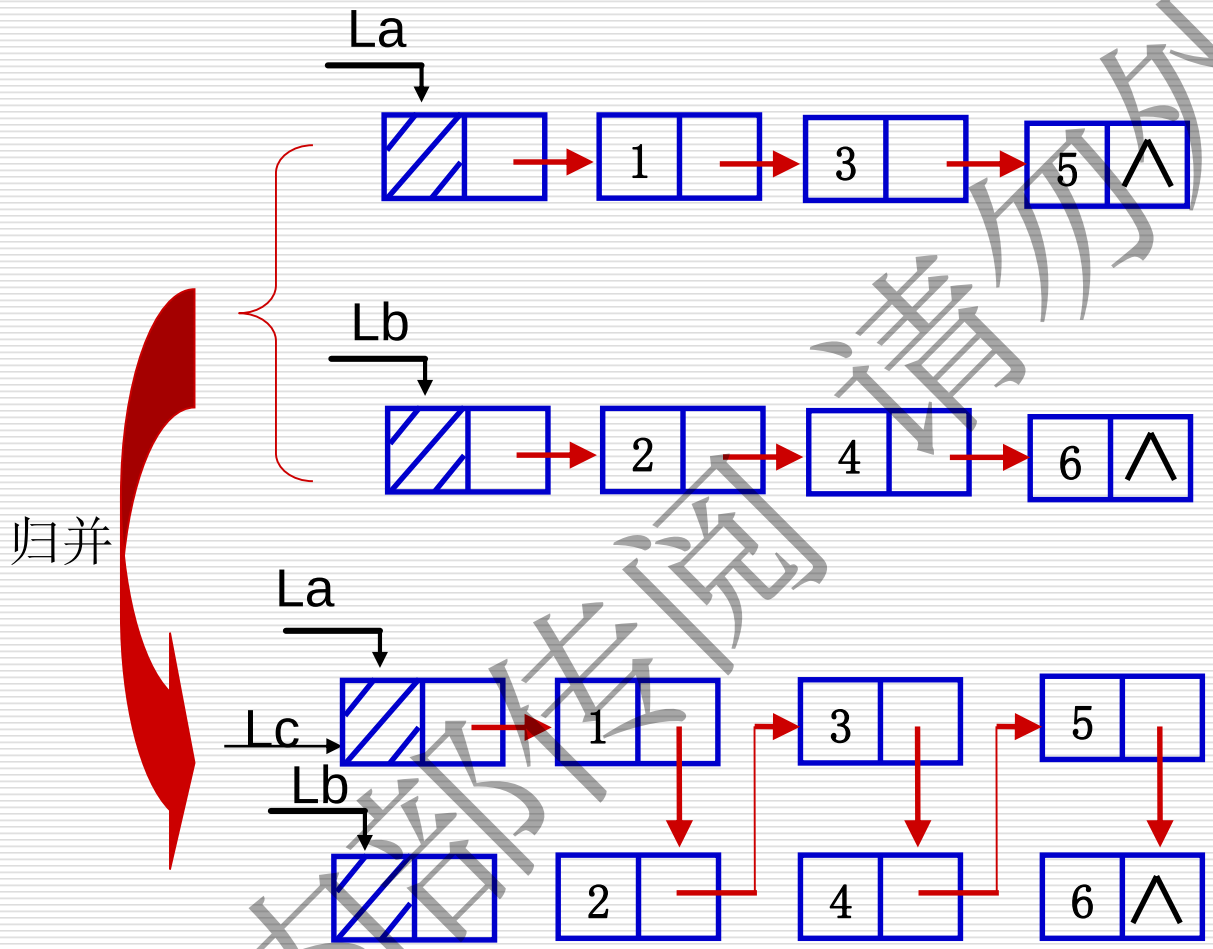
设两个指针pa,pb分别对La,Lb进行扫描，在扫描过程中按pa,pb所指结点的大小，将其插入到Lc的表尾。

# 线性链表归并操作算法（直接对链表进行操作）

```
void MergeList_L(LinkList &La, LinkList &Lb, LinkList &Lc) {
//已知单链线性表La和Lb的元素按值非递减排列
//归并La和Lb得到新的单链线性表Lc,Lc的元素也按值非递减排列。
pa=La->next; pb=Lb->next;
Lc=pc=La;  //用La的头结点作为Lc的头结点
while(pa && pb){
If (pa->data<=pb->data)
      {pc->next=pa;pc=pa;pa=pa->next;}
else {pc->next=pb;pc=pb;pb=pb->next;}
}
pc->next=pa?pa:pb;//插入剩余段
free(Lb);//释放Lb的头结点
}//MergeList_L
```

算法 2.12

# 线性链表归并操作图示



# 2.8 案例分析与实现

## 一、一元多项式的表示

数学上,一元多项式可按升幂写成：

$$P_n(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n$$

$$Q_m(x) = q_0 + q_1 x + q_2 x^2 + \cdots + q_m x^m$$

为了用计算机实现多项式运算，可用一个由系数构成的线性表来表示一元多项式：

$P=(p_0, p_1, p_2, \ldots p_n)$

$Q=(q_0, q_1, q_2, \ldots q_m)$

设m<n，则两个多项式相加的结果可用线性表R表示：

$R=(p_0+q_0, p_1+q_1, \ldots, p_m+q_m, p_{m+1}, \ldots, p_n)$

但应用中一元多项式往往幂次很高，且有大量系数为零。

例：　　$S(x) = 1 + 3x^{10000} + 2x^{20000}$

如果用系数构成的线性表表示该多项式，就要用一长度为**20001**的线性表表示，可表中只有三个非零项，这显然很浪费内存空间。可采用另一种方式，即用非**0**项（系数，指数）构成的线性表表示一元多项式，

**S= ((1,0), (3,10000), (2,20000))**

以下用非0项（系数，指数）构成的线性表表示一元多项式。用这种方式表示一元多项式时，一元多项式运算经常要对线性表进行插入删除操作，所以采用线性链表存储结构它们。

# 一元多项式链式存储结构

## 1）元素的类型定义

```
typedef struct PNode{//项的表示
float   coef ;              //系数
int     expn;               //指数
struct PNode *next;  //指针
}PNode;
```


## 2）一元多项式链表的类型定义

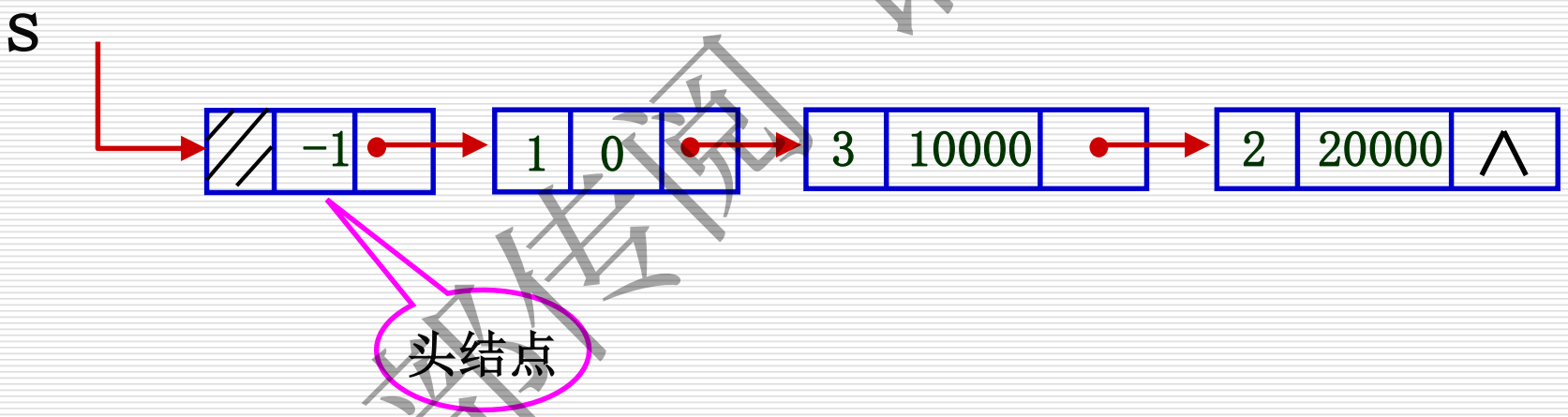
```
typedef  PNode  *Polynomial;
```

//用带表头结点的线性链表表示多项式，

//为类型重命名是为增加算法的可读性

## 3） 一元多项式链式存储结构图示

例： $S(x) = 1 + 3x^{10000} + 2x^{20000}$



# 一元多项式的相加算法

## 1 一元多项式的相加

一元多项式相加运算规则：指数相同的项系数相加。

例：求两多项式的和多项式

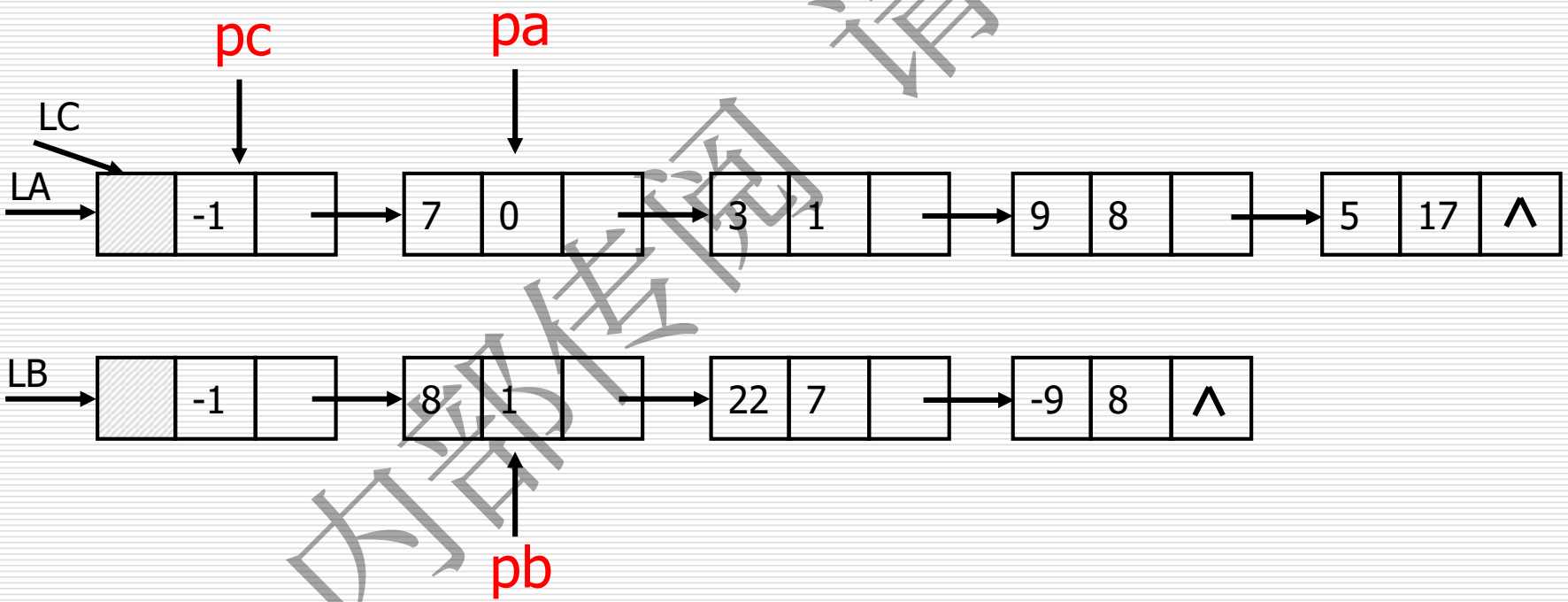
$$A(x) = 7 + 3x + 9x^{8} + 5x^{17}$$

$$B(x) = 8x + 22x^{7} - 9x^{8}$$

$$C(x) = A(x) + B(x) = 7 + 11x + 22x^{7} + 5x^{17}$$

$A_{17}(x)=7+3x+9x^8+5x^{17}$

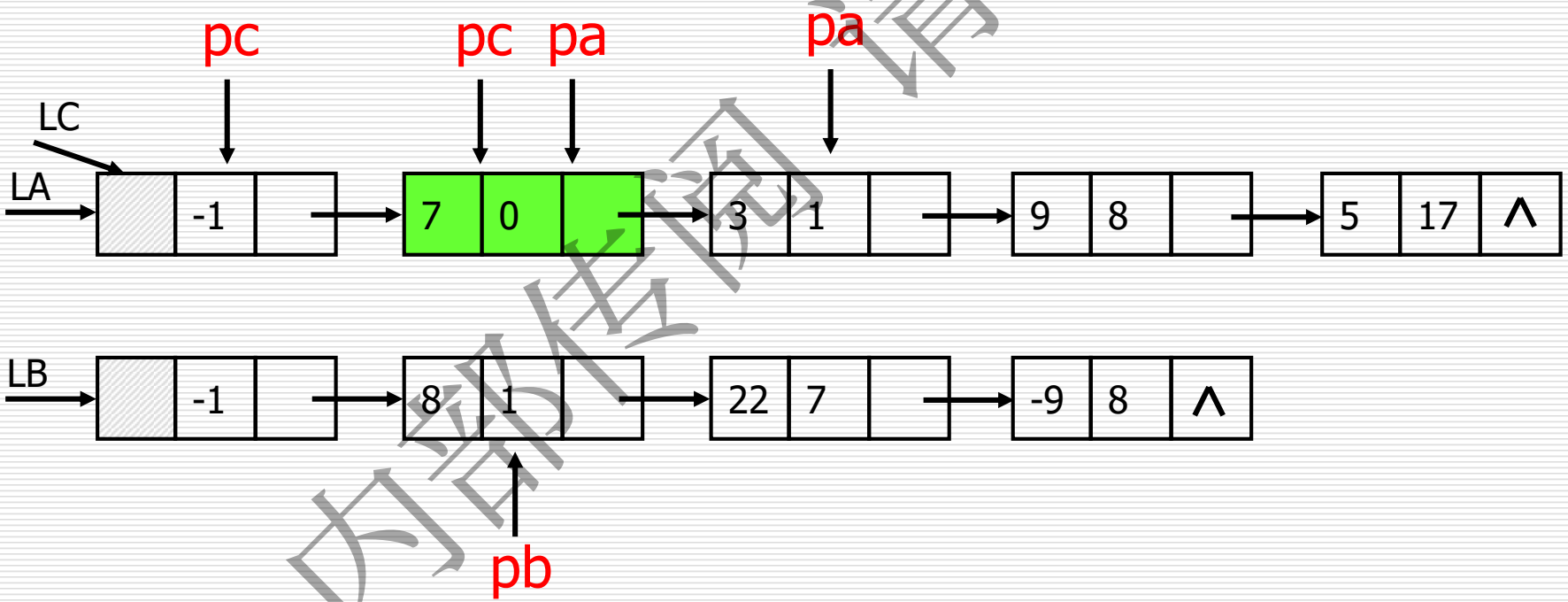
$B_8(x)=8x+22x^7-9x^8$



# 多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

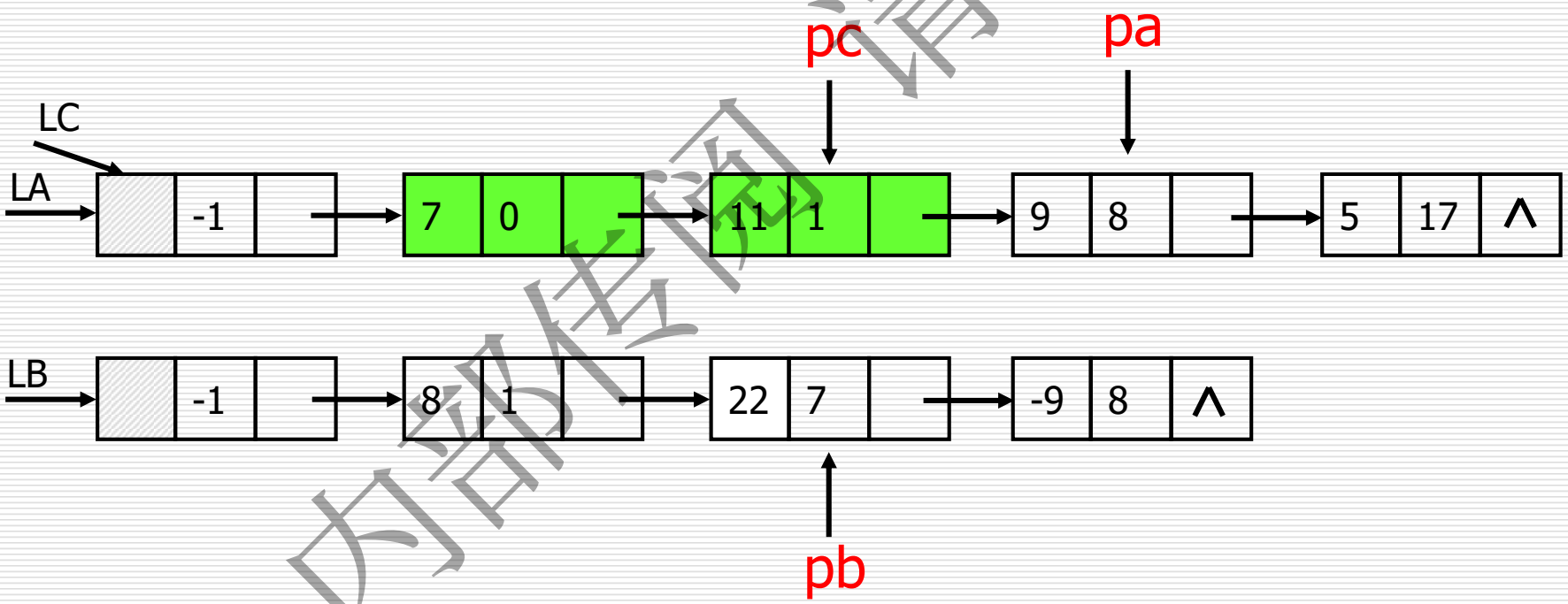
$B_8(x)=8x+22x^7-9x^8$



# 多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

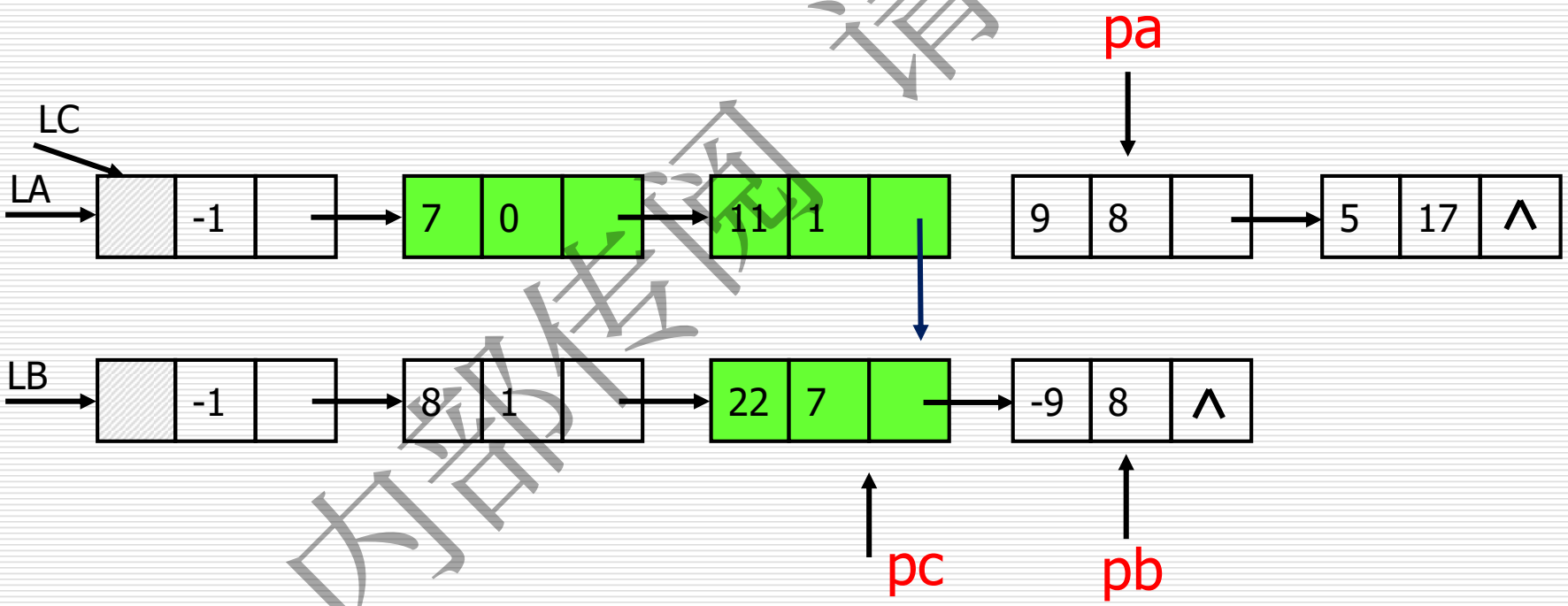
$B_8(x)=8x+22x^7-9x^8$



# 多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

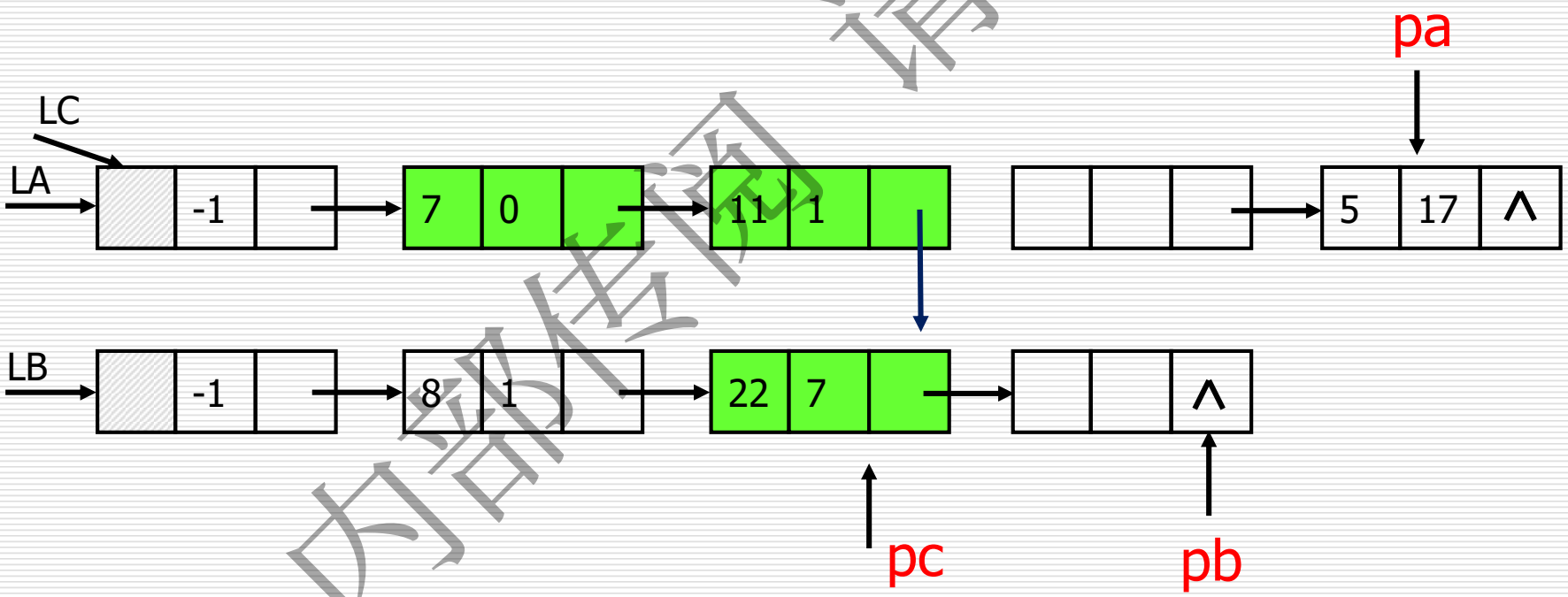
$B_8(x)=8x+22x^7-9x^8$



# 多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

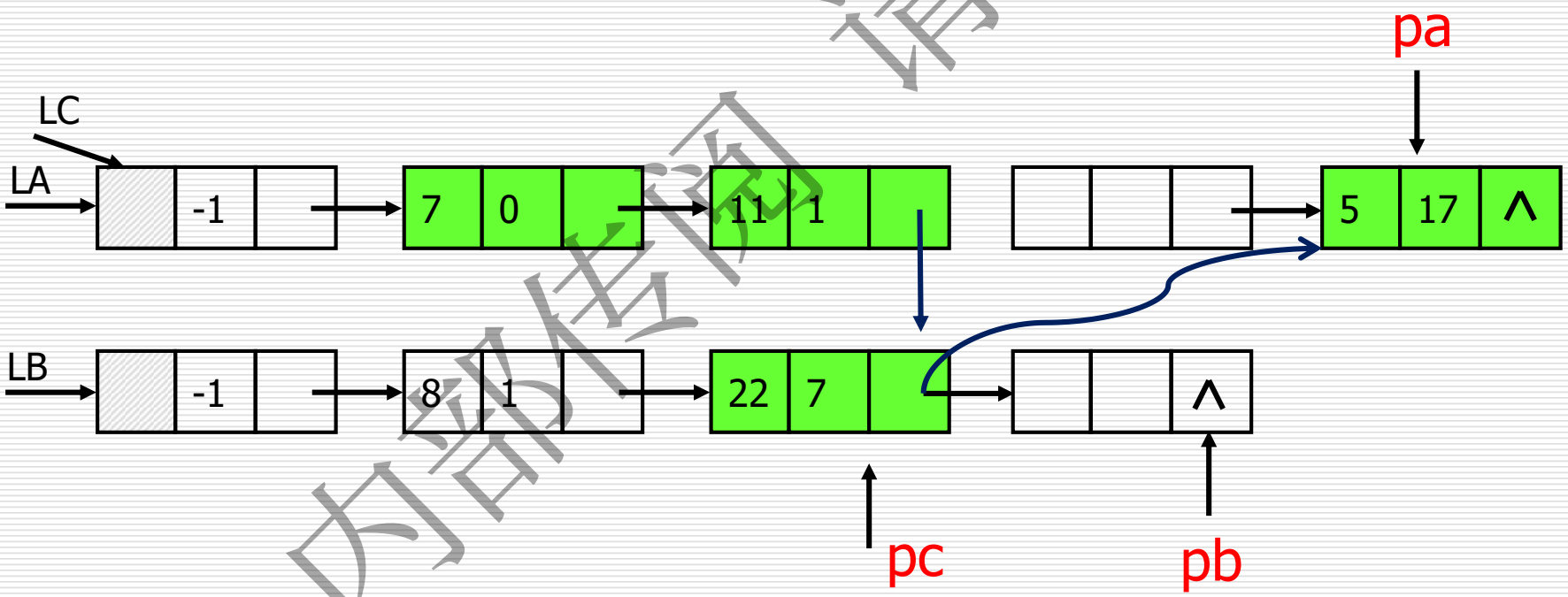
$B_8(x)=8x+22x^7-9x^8$



# 多项式相加

$A_{17}(x)=7+3x+9x^8+5x^{17}$

$B_8(x)=8x+22x^7-9x^8$



# 一元多项式加法算法主要步骤

分别对两个链表La 、Lb进行扫描，设pa 、pb 分别指向线性链表La 、 Lb的当前进行比较的某个结点，比较两个结点的指数项，有下列三种情况：

（1）if(pa->expn < pb->expn )

pa所指结点应为和多项式中的结点；

pc->next=pa;

pc=pa;

pa=pa->next;

（2）else if(pa->expn > pb->expn )

pb所指结点应为和多项式中的结点；

pc->next=pb;

pc=pb;

pb=pb->next;

# 一元多项式加法算法主要步骤

**（3）else if(pa->expn = =pb->expn )**

将两个结点系数域中的值相加。如果相加的和为零，则删除**pa**和**qb**所指结点；如果相加的和不为零，则修改**pa**所指结点的系数值，同时释放**pb**所指结点；

```
{ if(pa->coef+pb->coef==0)
     {pa=pa->next;
     pb=pb->next;}
  else{ pa->coef=pa->coef+pb->coef;
        pc->next=pa;
        pc=pa;
        pa=pa->next;
        pb=pb->next;}
}
```

# 2.9 小结

- 本章学习了线性表的顺序存储结构——顺序表，链式存储结构——线性链表、循环链表、 双向链表，以及在这两种存储结构下如何实现线性表的基本操作。
- 不仅要从概念和方法上了解每一种数据结构的逻辑结构和基本操作，更重要的是要学习如何在计算机上实现，即如何在计算机上存储线性表，如何在计算机上实现线性表的操作。
- 我们要通过数据结构课程的学习，很好地理解各种存储结构是如何存储和表达数据对象的有关信息的，各种存储结构下操作的特点。为实际问题的程序设计打下坚实的基础。

# 习题选讲

**1.顺序表操作**

**（1）顺序表判空操作**

```
status ListEmpty_Sq(SqList L)
{
   if(L.length==0) return TRUE;
   else return FALSE;
}
```

**（2）顺序表求表长操作**

```
int ListLength_Sq(SqList L)
{
    return L.length;
}
```

**2.设顺序表va中的数据元素递增有序，试写一算法，将x插入到顺序表的适当位置上，以保持该表的有序性。**

```
解: Status Insert_SqList(SqList &va,int x)
{    //把x插入递增有序表va中
     if(va.length+1>va.listsize)
{ //当前存储空间已满，重新分配空间
        newbase=(ElemType*)realloc(L.elem,
    (L.listsize+LISTINCREMENT)*sizeof (ElemType));
        if (!newbase)exit(OVERFLOW);  //存储分配失败
        L. elem=newbase; //新基址
        L.listsize+=LISTINCREMENT; //增加存储容量
    }
```

```
va.length++;
  for(i=va.length-1;va.elem[i]>x&&i>=0;i--)
      va.elem[i+1]=va.elem[i];
      va.elem[i+1]=x;
 return OK;
}
//Insert_SqList
```

**3.** 试写一算法在带头结点的单链表结构上实现线性表操作**LENGTH（L）。**

解: int LENGTH(LinkList L)//求链表的长度

```
{
   for(k=0,p=L;p->next;p=p->next,k++);
   return k;
}//LENGTH
```

**4.** 试写一算法，在无头结点的动态链表上实现线性表操作INSERT(L,I,b)，并和在带头结点的动态单链表上实现相同操作的算法进行比较。

```
解: typedef struct LNode
    {   elemtype data;
        struct LNode *next;
     }LNode, *Linklist;

void Insert(LinkList &L, int i, Elemtype b)
{//在无头结点链表L的第i个元素之前插入元素b
   p=L;
   q=(LinkList)malloc(sizeof(LNode));
   q->data=b;
```

```
if(i==1)
  {
    q->next=p;
    L=q;   //插入在链表头部
   }
else
   {
    while(--i>1) p=p->next;  //p指向第i-1个元素
    q->next=p->next;
    p->next=q; //将q插入在第i个元素之前
   }
}//Insert
```

## 5. 试写一算法，对单链表实现就地逆置。

```
解: void Invertlist(LinkList head)
  { //将带头结点的单链表head就地逆置（利用头插法重构单链表）
      LinkList p,q;
      p=head->next;     //p指向链表的第一个结点
      head->next=NULL;      //断开头结点
      while(p!=NULL)
       { q=p->next;   //q记住下一个待插结点
         p-next=head->next;  //将p所指结点插到头结点后
         head->next=p;
         p=q;
       }
}//Invertlist
```

# 6. 约瑟夫环问题

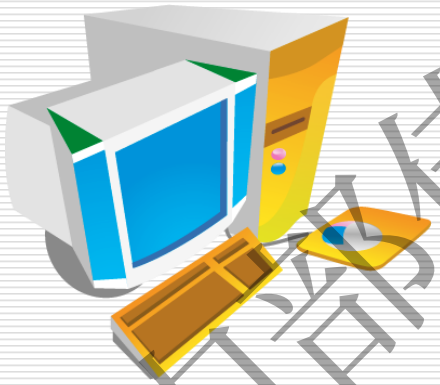
设有n个人围坐一圈，现从某个人开始报数，数到m的人出列，接着从出列的下一个人开始重新报数，数到m的人又出列，如此下去，直到所有人都出列为止。求最后一个出队列的人的序号。

```
#include <stdio.h>
#include <malloc.h>
#define NULL 0
typedef struct LNode{
  int data;
  struct LNode *next;
}LNode, *LinkList;
```

```
void main()
{ //不带头结点的循环链表解决约瑟夫环问题.
  LinkList L,p;
  printf("Please input the Number and the order:\n");
  scanf("%d,%d",&N,&k);//N代表总人数,k代表出列号.
   L=(LinkList)malloc(sizeof(LNode));
   L->data=1;
   L->next=L;//首先生成第一个结点
   for(i=N;i>1;i--)
     {  p=(LinkList)malloc(sizeof(LNode));
        p->data=i;
              p->next=L->next;
              L->next=p;
     }//建立循链表
```

```
p=L;
  while(p->next!=p)
  { for(i=1;i!=k-1;i++)
      p=p->next;
    printf("%d    ",p->next->data);
    p->next=p->next->next;
          p=p->next;
  }
  printf("\n");
  printf("The last one is:%d\n",p->data);
}
```

# 第二章　结束



***Thank you!***