第2章 表

2.1 ADT表
2.2 用数组实现ADT表
2.3 用指针实现ADT表
2.4 用间接寻址方法实现ADT表
2.5 用游标实现ADT表
2.6 循环链表
2.7 双链表
2.8 表的搜索游标
2.9 表的应用——Josephus排列问题

学习要点:

- 理解表是由同一类型的元素组成的有限序列的概念。
- 熟悉定义在抽象数据类型表上的基本运算。
- 掌握实现抽象数据类型的一般步骤。
- 掌握用数组实现表的步骤和方法。
- 掌握用指针实现表的步骤和方法。
- 掌握用间接寻址技术实现表的步骤和方法。
- 掌握用游标实现表的步骤和方法。
- 掌握单循环链表的实现方法和步骤。
- 掌握双链表的实现方法和步骤。
- 熟悉表的搜索游标的概念和实现方法。

2.1 ADT表(List)

2.1.1 ADT表的数据模型

表是由n(n≥0)个同一类型的元素a(1)，a(2)，…，a(n)组成的有限序列。

2.1.2 有关的概念与术语

- 表的长度（Length）：表的元素的个数即数据模型中的n。
- 空（Empty）表：n=0的表。
- 表中元素(结点)的位置（Position）：当n≥1时，说k是表中第k个元素a(k)的位置，k=1，2，…，n。
- 表中元素(结点)的前驱：当n>1时，说a(k)是a(k+1)的前驱(k=1，2，…，n-1)。
- 表中元素(结点)的后继：当n>1时，说a(k+1)是a(k)的后继(k=1，2，…，n-1)。

2.1 ADT表（List）

2.1.3 ADT表的逻辑特征

- 非空表有且仅有一个开始元素a(1)，它没有前驱。当n>1时，它有一个后继a(2)。
- 非空表有且仅有一个结束元素a(n)，它没有后继。当n>1时，它有一个前驱a(n-1)。
- 当n>2时，表的其余元素a(k)(2≤k≤n-1)都有一个前驱和一个后继。
- 表中元素按其位置的顺序关系是它们之间的逻辑关系 。

2.1 ADT表（List）

2.1.4 ADT表上定义的常用的基本运算

(1) ListEmpty(L):

(2) ListLength(L):

(3) ListLocate(x,L): 返回表中元素x位置

(4) ListRetrieve(k,L): 获取表中位置k处的元素

(5) Insert(k,x,L): 在表的位置k之后插入元素x

(6) Delete(k,L): 从表中删除位置k处的元素

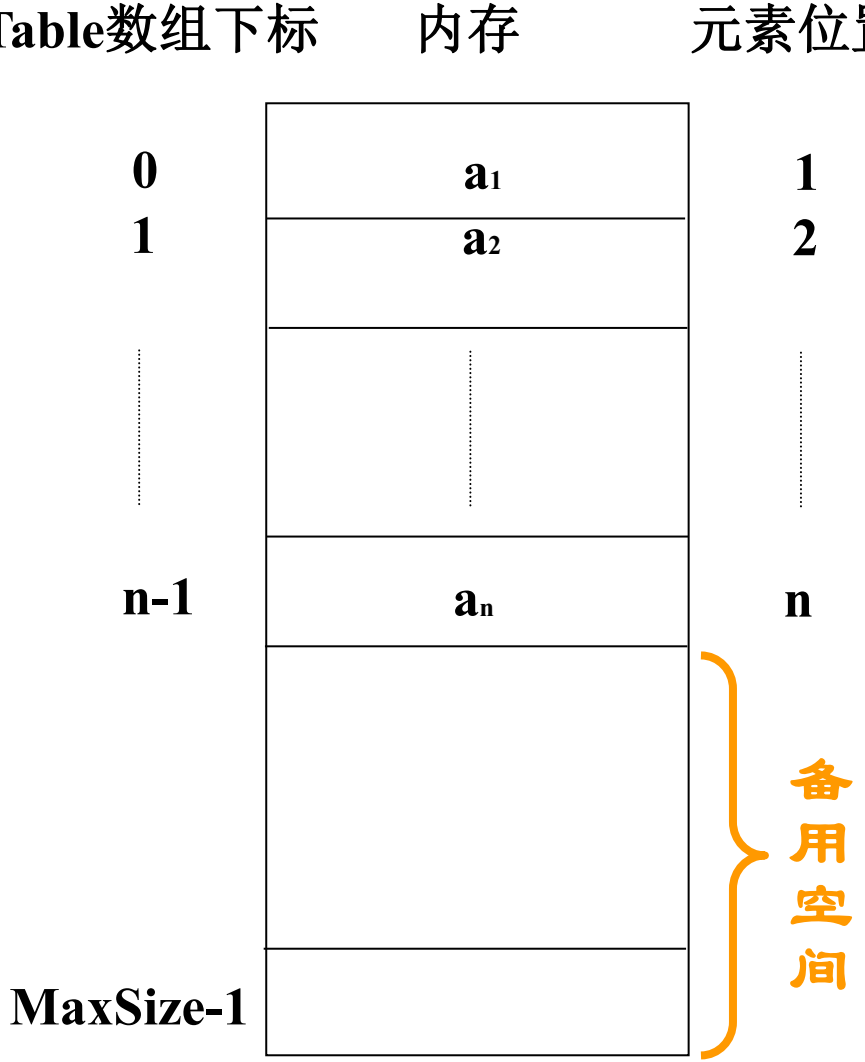
(7) PrintList(L):

返回章目录

2.2 用数组(data)实现ADT表（List）

2.2.1 用数组实现的ADT表的特征数据及其类型

- 表的元素的类型：为了适应表元素类型的变化，应将表类型**List**定义为一个结构。在该结构中，用**ListItem**表示用户指定的元素类型。
- 表的长度:**n(**约定**n=0**为空表**)**
- 数组能容纳的表的最大长度:**MaxSize**
- 约定数组下标为**k-1**的分量存放表的第**k**个元素，**k=1,2,3,…,n**。其结构如下图


Table数组下标
内存
元素位置
0
1
n-1
MaxSize-1
a1
a2
an
1
2
n
备用空间

2.2.2用数组实现的ADT表的结构定义

```
typedef  struct alist   *List;
typedef  struct alist {
    int  n;                  /* 表长，当表为空时，n=0*/
    int  maxsize;            /*表示线性表的最大长度 */
    ListItem   table;        /*表元素数组*/
} Alist;
```

ADT表的7个基本运算及初始化运算的接口：

```
List ListInit (int size);
int ListEmpty(List L);
int ListLength(List L);
int ListLocate (ListItem x, List L);
ListItem ListRetrieve(int k, List L);
void ListInsert (int k, ListItem x, List L);
ListItem ListDelete(int k, List L);
void  PrintList(List  L);
```

2.2.3 ADT表（List）的基本运算的实现与分析

1、初始化函数

分配大小为size的空间给表数组table，并返回初始化为空的表。

```
 List ListInit (int size)
{
    List  L;
    L= (List) malloc(sizeof *L);
    L->table= (ListItem) malloc (size*sizeof(ListItem));    /*分配空间*/
    L->maxsize=size;
    L->n=0;                          /*将当前线性表长度置0*/
    return L;                        /*成功，返回L*/
}
```

时间复杂性:O(1)

2.2.3 ADT表（List）的基本运算的实现与分析

2、判断表是否为空及求表长函数

（1）判断表L是否为空

```
int ListEmpty(List L)
{
    return   L->n==0;
}
```

（2）求表长

```
int ListLength(List L)
{
   return  L->n;
}
```

以上两个程序的时间复杂性均为：O(1)

2.2.3 ADT表（List）的基本运算的实现与分析

3、元素x定位函数

返回元素x在表中的位置，当元素x不在表中时返回0。

```
int ListLocate (ListItem x, List L)
{
  int  i;
  for (i=0; i< L->n; i++)
      if (L->table[i]==x)    return  ++i;
  return 0;
}
```

最坏情况时间复杂性:O(n)

2.2.3 ADT表（List）的基本运算的实现与分析

4、获取线性表L中的某个数据元素内容的函数

```
ListItem ListRetrieve(int k, List L)
{
  if (k<1 || k>L->n) Error("out of bounds");
  /*判断k值是否合理，若不合理，ERROR*/
  return L->table[k-1];
}
```

时间复杂性:O(1)

2.2.3 ADT表（List）的基本运算的实现与分析

5、在位置k后插入元素x: void ListInsert(int k, ListItem x, List L);

- 插入定义：线性表的插入是指在第k（$0 \le k \le n$）个元素之后插入一个新的数据元素x，使长度为n的线性表

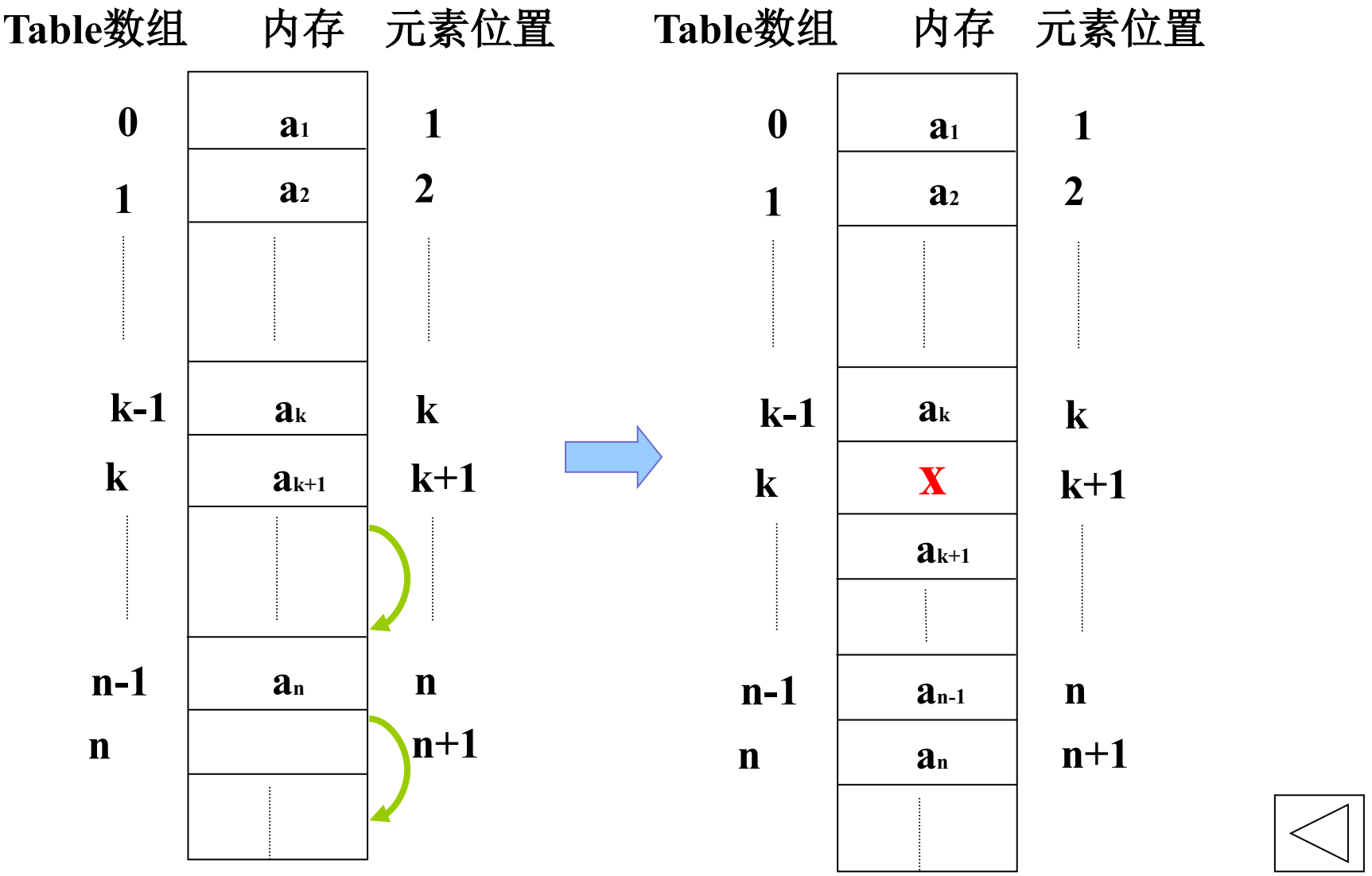
$$(a_1, a_2, \ldots\ldots, a_k, a_{k+1}, \ldots\ldots a_n)$$

变成长度为n+1的线性表

$$(a_1, a_2, \ldots\ldots, a_k, x, a_{k+1}, \ldots\ldots a_n)$$

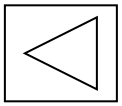
需将第k+1至第n共（n-k）个元素后移

- 图示
- 算法
- 复杂性分析


Table数组
内存
元素位置
0
1
k-1
k
n-1
n
a1
a2
ak
ak+1
an
1
2
k
k+1
n
n+1
Table数组
内存
元素位置
0
1
k-1
k
n-1
n
a1
a2
ak
X
ak+1
an-1
an
1
2
k
k+1
n
n+1

- 算法：

```
void ListInsert (int k, ListItem x, List L)
{
  int  i;
  if (k<0||k>L->n) Error("out of bounds"); /*检查k是否合理*/
  if (L->n==L->maxsize) Error("out of memory");
    /*检查是否有剩余空间*/
  for (i=L->n-1;i>=k;i--)
    L->table[i+1]=L->table[i];
   /*将线性表第k个元素之后的所有元素向后移动
  L->table[k]=x;
    /*将新元素的内容放入线性表的第k+1个位置*/
  L->n++;
}
```



- 算法时间复杂度T(n)
  - 最坏情况时间复杂性：O(n)
  - 平均情况时间复杂性：设P_k是在第k个元素之后插入一个元素的概率，则在长度为n的线性表中插入一个元素时，所需移动的元素次数的平均次数为：

$$E_{IN} = \sum_{k=0}^{n} P_k (n-k)$$

$$若认为 P_k = \frac{1}{n+1}$$

$$则 E_{IN} = \frac{1}{n+1} \sum_{k=0}^{n} (n-k) = \frac{n}{2}$$

$$\therefore T(n) = O(n)$$

2.2.3 ADT表（List）的基本运算的实现与分析

6、删除位置k处的元素给x: ListItem ListDelete(int k, List L);

- 删除定义：线性表的删除是指将第k（1≤k ≤ n）个元素删除，使长度为n的线性表

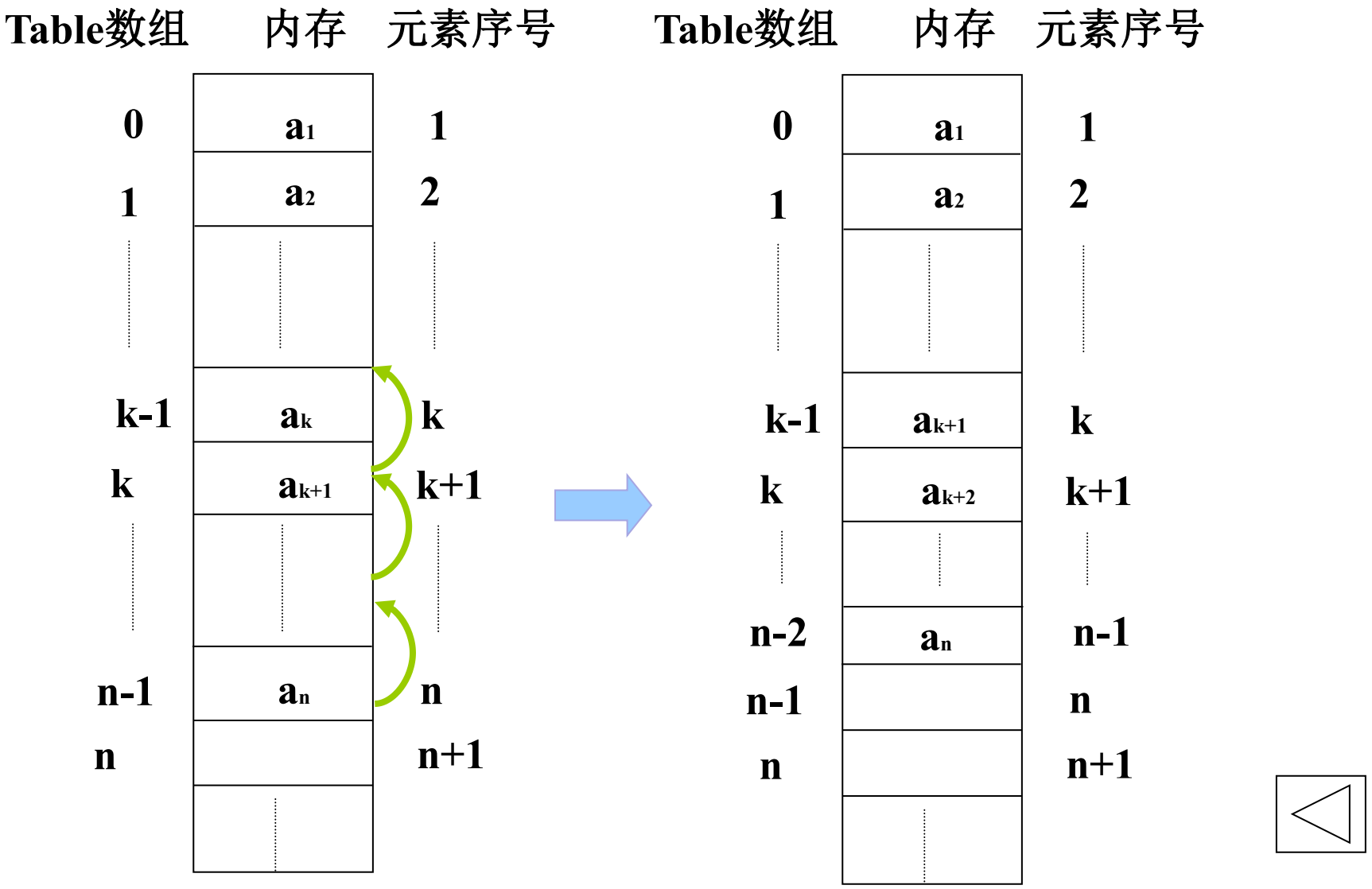
$$(a_1, a_2, \ldots\ldots, a_{k-1}, a_k, a_{k+1}, \ldots\ldots a_n)$$

变成长度为n-1的线性表

$$(a_1, a_2, \ldots\ldots, a_{k-1}, a_{k+1}, \ldots\ldots a_n)$$

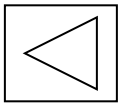
需将第k+1至第n共（n-k）个元素前移

- 图示
- 算法
- 复杂性分析


Table数组
内存
元素序号
0
1
k-1
k
n-1
n
a1
a2
ak
ak+1
an
1
2
k
k+1
n
n+1
Table数组
内存
元素序号
0
1
k-1
k
n-2
n-1
n
a1
a2
ak+1
ak+2
an
1
2
k
k+1
n-1
n
n+1

- 算法：

```
ListItem ListDelete (int k, List L)
{
  int  i; ListItem x;
  if (k<1||k>L->n) Error(“out of bounds”);;
  x=L->table[k-1];
  for (i=k; i<L->n; i++)
    L->table[i-1]=L->table[i];
  L->n--;
  return x;
}
```



- 算法时间复杂度T(n)
  - 最坏情况时间复杂性：O(n)
  - 平均情况时间复杂性：设Qk是删除第k个元素的概率，则在长度为n的线性表中删除一个元素时，所需移动的元素次数的平均次数为：

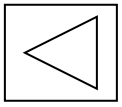
$$E_{DE}=\sum_{k=1}^{n}Q_k(n-k)$$

$$若认为Q_k=\frac{1}{n}$$

$$则E_{DE}=\frac{1}{n}\sum_{k=1}^{n}(n-k)=\frac{n-1}{2}$$

$$\therefore T(n)=O(n)$$

故在顺序表中插入或删除一个元素时，平均移动表中约一半的元素，当n很大时，效率很低



2.2.3 ADT表（List）的基本运算的实现与分析

7、输出顺序表中所有元素的运算

```
void  PrintList(List  L)
{
  int  i;
  for(i=0;i<L->n;i++)
     ItemShow(L->table[i]);   /*此函数用以输出元素*/
}
```

时间复杂性：O(n)

2.2 用数组(data)实现ADT表（List）

2.2.4 用数组(data)实现ADT表（List）的优缺点

- 优点
  - 逻辑相邻，物理相邻=>无须为表示表中元素之间的逻辑关系而增加额外的存储空间
  - 可随机存取任一元素
  - 存储空间使用紧凑
- 缺点
  - 插入、删除操作需要移动大量的元素
  - 预先分配空间需按最大空间分配，利用不充分
  - 表容量难以扩充

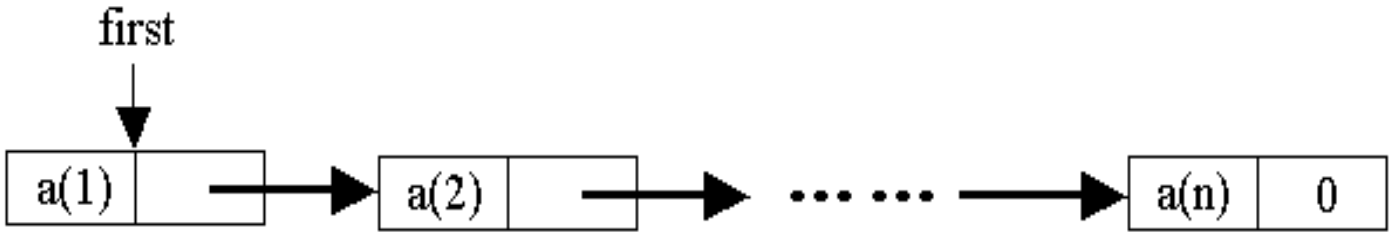
返回章目录

2.3 用指针实现ADT表（List）

2.3.0 用指针实现ADT表动因和构想

动因：用数组实现表存在两个缺点。

构想：表的每一个元素存放在随时可向操作系统申请到的单元（结点）内，前后结点靠一个指针来链接（单链）。结构如下图：

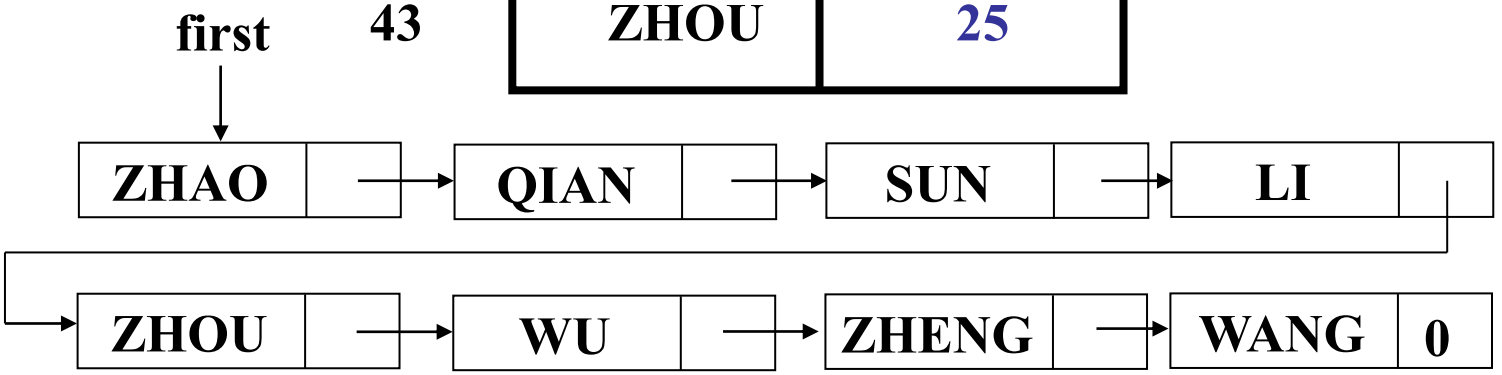


例 线性表 (ZHAO,QIAN,SUN,LI,ZHOU,WU,ZHENG,WANG)

第一个元素指针

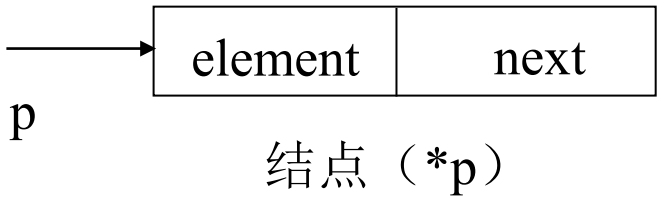
first
31

存储地址	数据域	指针域
1	LI	43
7	QIAN	13
13	SUN	1
19	WANG	0
25	WU	37
31	ZHAO	7
37	ZHENG	19
43	ZHOU	25



2.3.2 单链表的结点结构说明

```
typedef struct node * link;
typedef struct node {
    ListItem element;
    link next;
}Node;
```




```
link NewNode( ){
   link p;
   if(p=(link)malloc(sizeof(Node)))=0)
      Error("Exhausted memory.");
   else return p;
}
```

用指针实现表的结构List如下：

```
typedef  struct llist  *List;
typedef  struct llist
{
  link  first;
}Llist;
```

表List的数据成员first是指向表中第一个元素的指针，当表为空时first指针是空指针。

2.3.3 单链表的基本运算

1、创建一个空表

```
List  ListInit()
{
  List   L= (List) malloc(sizeof *L);
  L->first=0;
  return  L;
}
```

2、判断当前表L是否为空表

```
int  ListEmpty(List  L)
{
  return   L->first==0;
}
```

3、求表长函数

```
int ListLength(List L)
{
  int len=0;
  link  p;
  p=L->first;
  while(p)  /*通过对表L进行线性扫描计算*/
{
    len++;
    p=p->next;
 }
  return  len;
}
```

时间复杂性为：O(n)

4、获取链表L中的某个数据元素的内容

```
ListItem  ListRetrieve(int  k,List L)
{
  int  i;
  link  p;
  if(k<1)  Error(“out of bounds”);
  p=L->first;
  i=1;
  while(i<k && p)
   {
       p=p->next;
      i++;
  }
 return  p->element;
}
```

时间复杂性为：O(k)

5、定位元素x (查找运算)

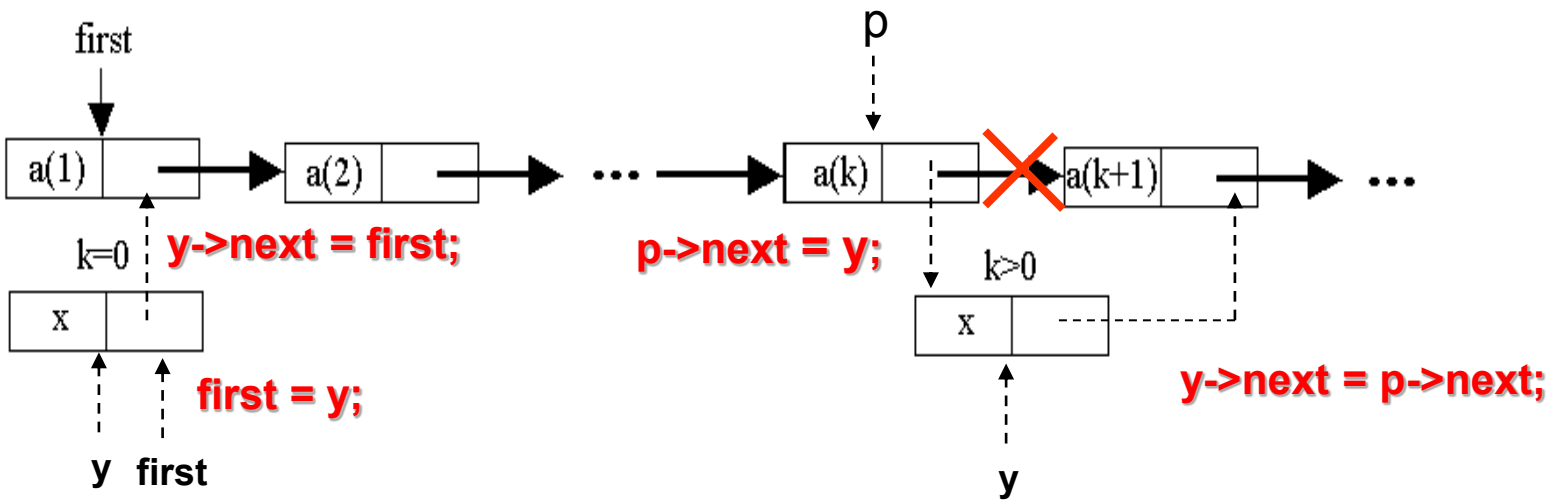
查找单链表中是否存在元素x，若有则返回元素x的位置；否则返回0；

```
int ListLocate(ListItem x, List  L)
{
 int i=1;
 link p;
 p=L->first;
 while(p&&p->element!=x)
 {
   p=p->next;
   i++;
 }
 return  p? i:0;  //如果p为空，说明x不存在返回0；否则返回其位置i
}
```

6、在位置k后插入元素x

函数的运算步骤是：先扫描链表找到插入位置p，然后建立一个存储待插入元素x的新结点y，再将结点y插入到结点p之后。

插入的示意图如下：



```
void  ListInsert(int k, ListItem x, List L)
{
   link p, y;
   int i;
   if (k < 0)    Error("out of bounds");
   p = L->first;  //找插入位置
   for  ( i= 1; i < k && p; i++)
       p = p->next;
   y = NewNode( );
   y->element = x;
   if (k) { y->next = p->next;   // 在位置p处插入
            p->next = y; }
   else {   y->next = first;
             first = y; } //在表首插入需要特殊处理（若引入表头结点则可纳入
                                                  一般情况）
}
```

算法的主要计算时间用于寻找正确的插入位置，故其时间复杂性为O(k)

7、删除位置k处的元素

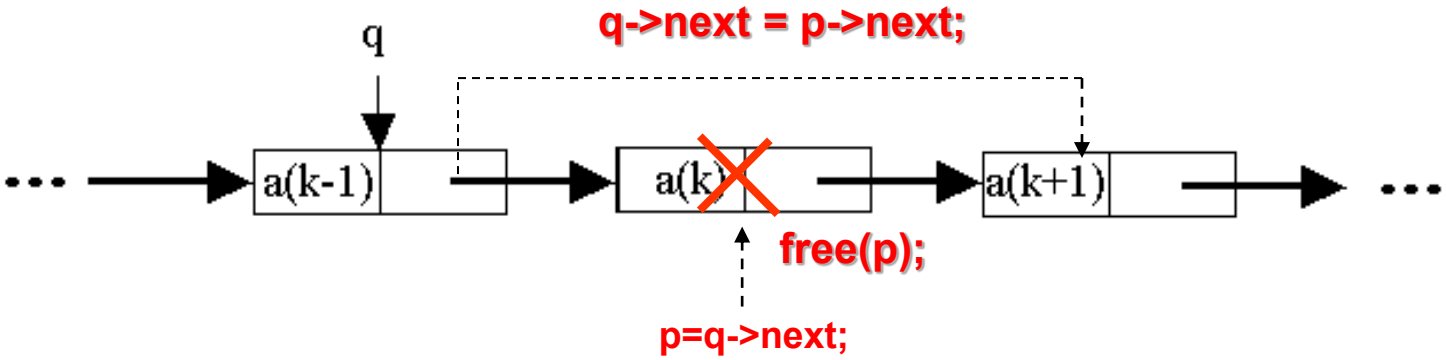
函数的运算步骤是：依次处理以下3种情况。

（ｉ）k<1或链表为空 → 给出越界信息；

（ii）删除的是表首元素，即k=1 →

直接修改表首指针**first**，删除表首元素

（iii）删除的是非表首元素，即k>1。其删除的示意图如下



```
ListItem ListDelete(int k, List L)
{  link  p, q;
   ListItem  x;
   int i;
   if (k < 1 || ! L->first)    Error("OutOfBounds" );
   p = L->first;
   if (k == 1) // 删除表首元素需要特殊处理
       L->first = p>next;
   else {// 找表中第k-1个元素所在结点q
     q =L-> first;
     for ( i = 1; i < k - 1 && q; i++)
          q = q->next;
     p = q->next;  //让p指向 第k个元素所在结点
     q->next = p->next;} // 删除结点p
     x = p->element; // 第k个元素存入x并释放结点p
     free( p );
     return   x;
  }
```

算法主要时间用于寻找待删除元素所在结点，故其所需时间为0(k)

8、输出链表中所有元素:

```
void  PrintList (List  L)
{
   link  p;
   for (p = L->first; p; p = p->next)
       ItemShow(p->element);
  }
```

时间复杂性: **O(n)**

2.3 用指针实现ADT表(List)

2.3.4 用指针实现ADT表（List）的优缺点

- 优点：

避免了数组要求连续的单元存储元素的缺点，因而在执行插入或删除运算时，不再需要移动元素来腾出空间或填补空缺。（当元素的粒度很大时，移动元素是很费时的。）

- 缺点：

需要在每个单元中设置指针来表示表中元素之间的逻辑关系，因而增加了额外的存储空间，为获得上述优点付出代价。

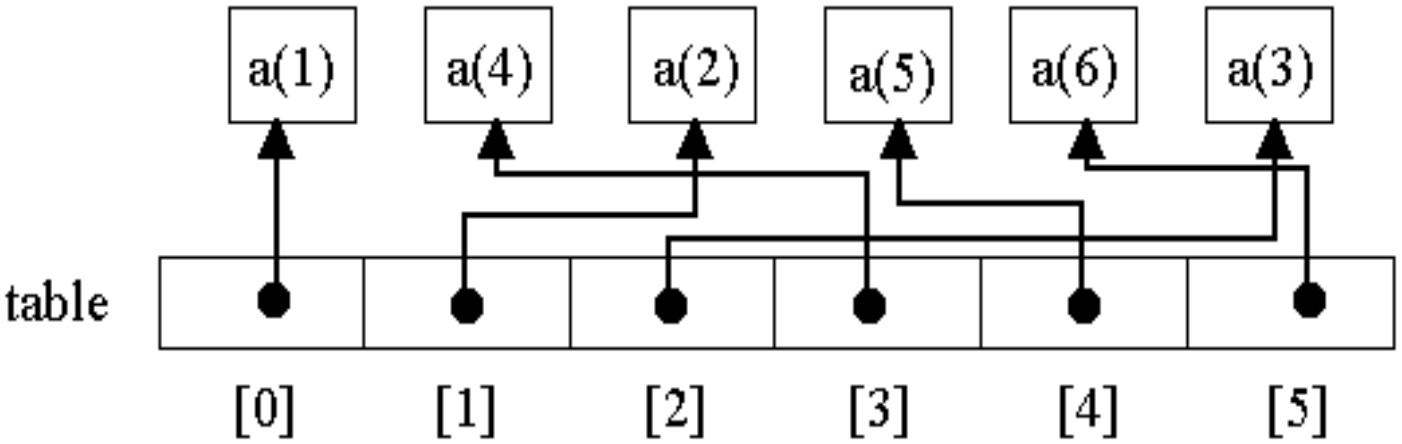
返回章目录

2.4 用间接寻址方法实现表

2.4.1 用间接寻址方法实现表的动因和构想

动因：综合数组和指针实现两者的优点。

构想：将数组和指针两种实现方式结合起来，让数组中原来存储元素的地方改为存储指向元素的指针。其结构图如下:



2.4 用间接寻址方法实现表

2.4.2 用间接寻址实现ADT表（List）的优缺点

- 优点：
  - (1)与用数组实现一样，可以方便地随机存取表中任一位置的元素。
  - (2)与用指针实现一样，在执行插入或删除运算时，不需要移动元素来腾出空间或填补空缺。（当元素的粒度很大时，移动元素是很费时的。）
- 缺点：
  - (1)与用数组实现一样，需要预先确定table的大小。当表长变化很大时，这比较难。
  - (2)与用指针实现一样，需要额外的存储空间，即额外的指针数组table。

返回章目录

2.5 用游标实现表

2.5.1 用游标实现表的动因和构想

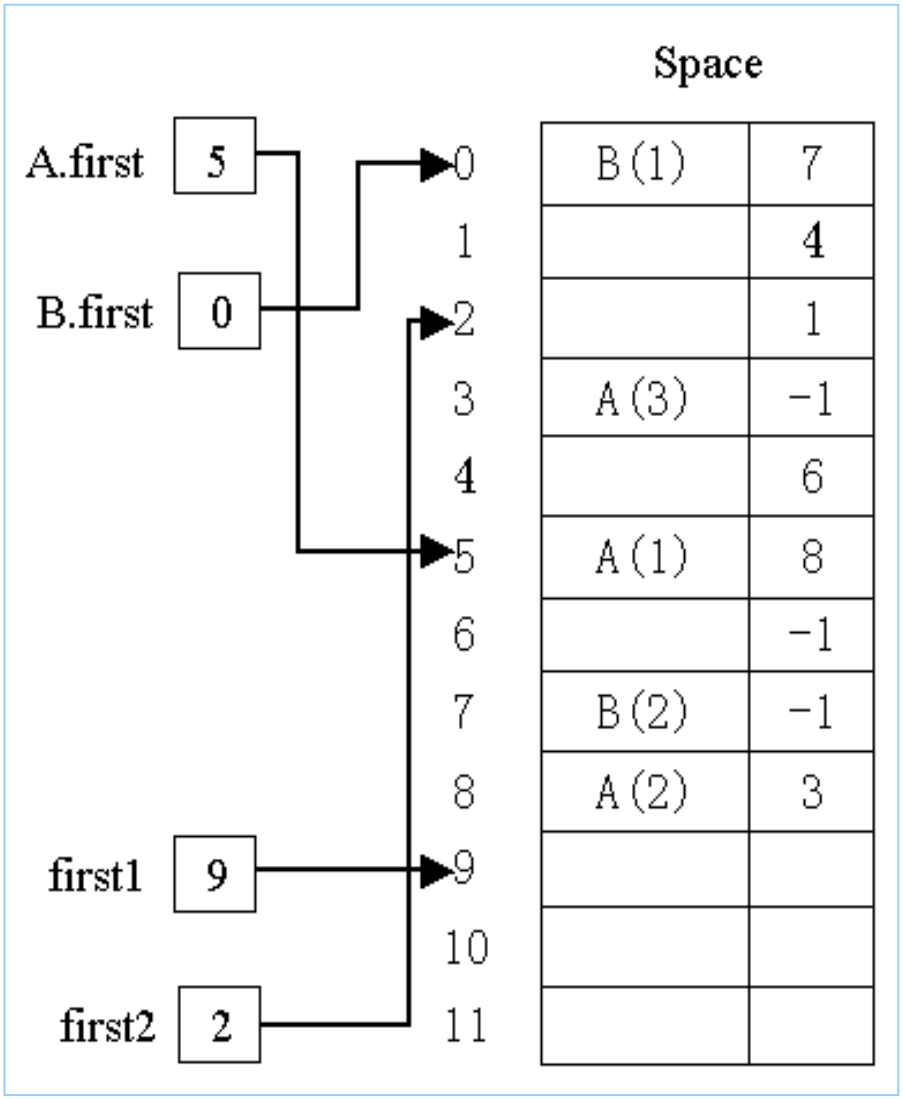
- 动因：对于有多个同类表的应用，希望通过自主调济内存资源，达到资源的更有效合理的利用。

- 构想：从操作系统申请一个较大的数组S，然后自主地支配S中的单元，在S中用游标模拟指针实现表，并让多个同类的表共享这个数组，如下页图。数组S[12]存放着相同类型的两个表A和B,其中表A含有3个元素；表B含有2个元素。

2.5 用游标实现表

first1指示S中尚未被使用的子段的起始单元；

first2指示S中被使用过、但目前处于闲的单元组成的一条可管理的链。让其中的单元优先于first1指示的子段中的单元满足应用的需要。



2.5 用游标实现表

2.5.2 用游标实现表的优缺点

- 优点:可实现多个同类的表共享同一片连续存储空间，给用户予资源调济的自主权。

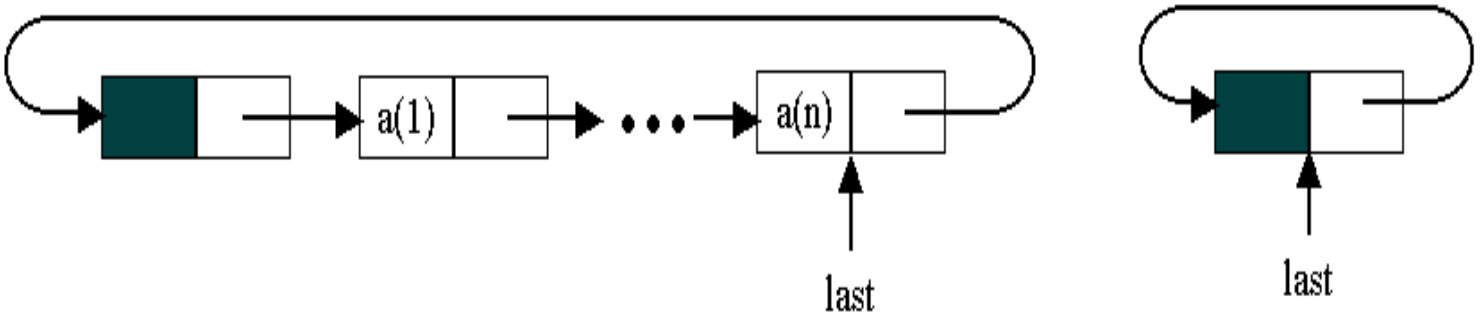
- 缺点:应该向操作系统申请多大的连续存储空间依赖于具体的应用，不容易把握。

返回章目录

2.6 用单循环链实现表

2.6.1 用单循环链实现表的动因和构想

- 动因：在用指针实现表时，表尾结点中的指针为空指针NULL。 人们希望把这个指针用起来，提高查找的效率。
- 构想：引入哨兵结点，并将尾结点的空指针改为指向哨兵结点的指针，使整个链表形成一个环。然后，用指向最后结点的指针表征我们的ADT表，结构如下图。



(a)非空表　　(b)空表

照此，找最后元素只需O(1),找首元素仍只需O(1)。

2.6 用单循环链实现表

2.6.2 用单循环链实现表的优点：

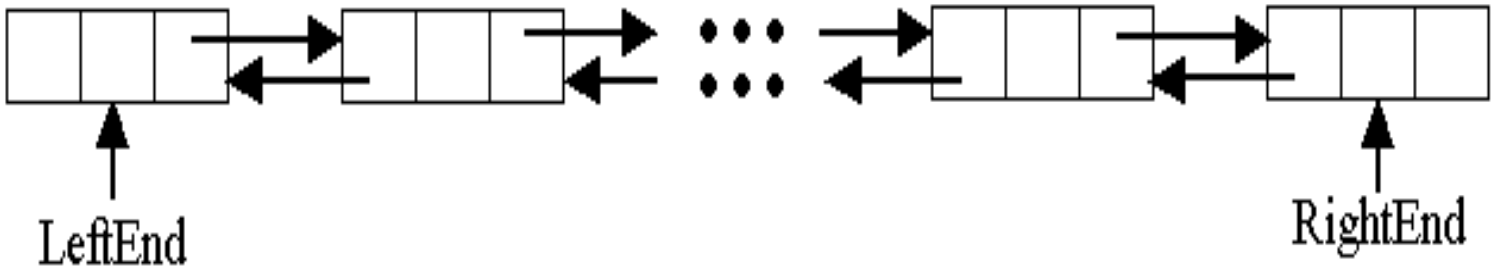
与单链表相比，可在O(1)的时间里找到表首元素和表尾元素。

返回章目录

2.7 用双链实现表

1、用双链实现表的动因和构想

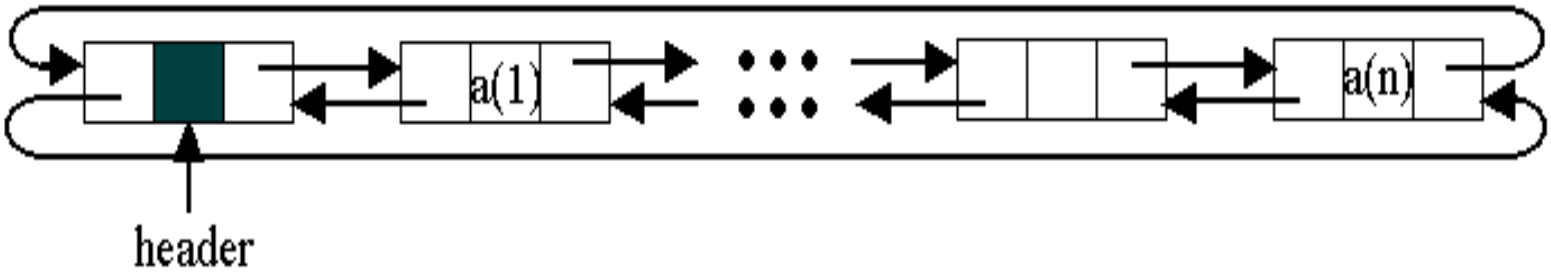
- 动因：无论在用指针实现表还是在用单循环链实现表时，对于表中任一元素x, 都不可能在O(1)时间里找到它的前驱。为了能在O(1)时间里既能找到前驱又能找到后继，提出了双链表。
- 构想：在用指针实现表的每一个结点中增加一个指向前驱的指针。结构如下图



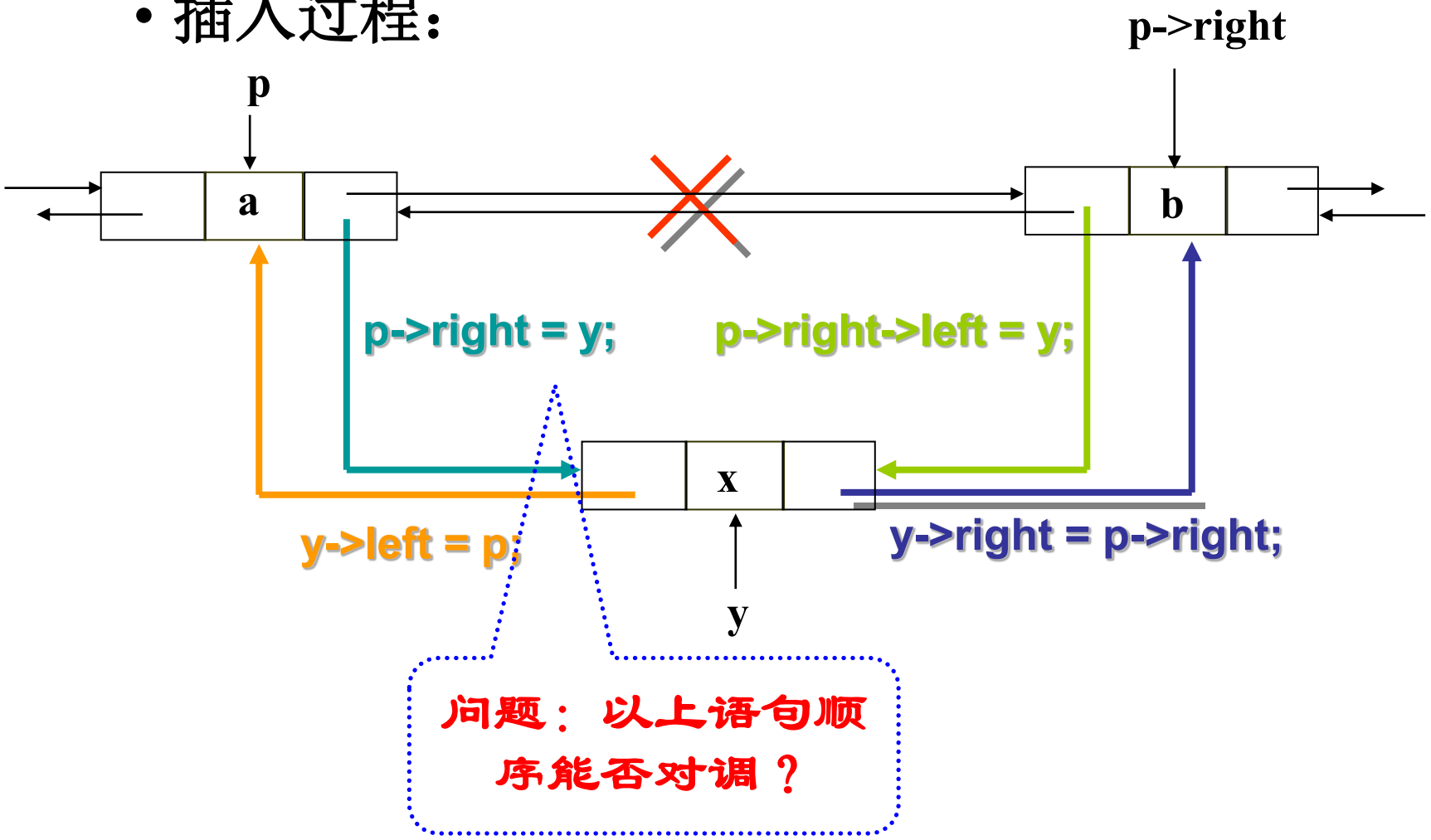
2.7 用双链实现表

2、用双循环链实现表的动因和构想

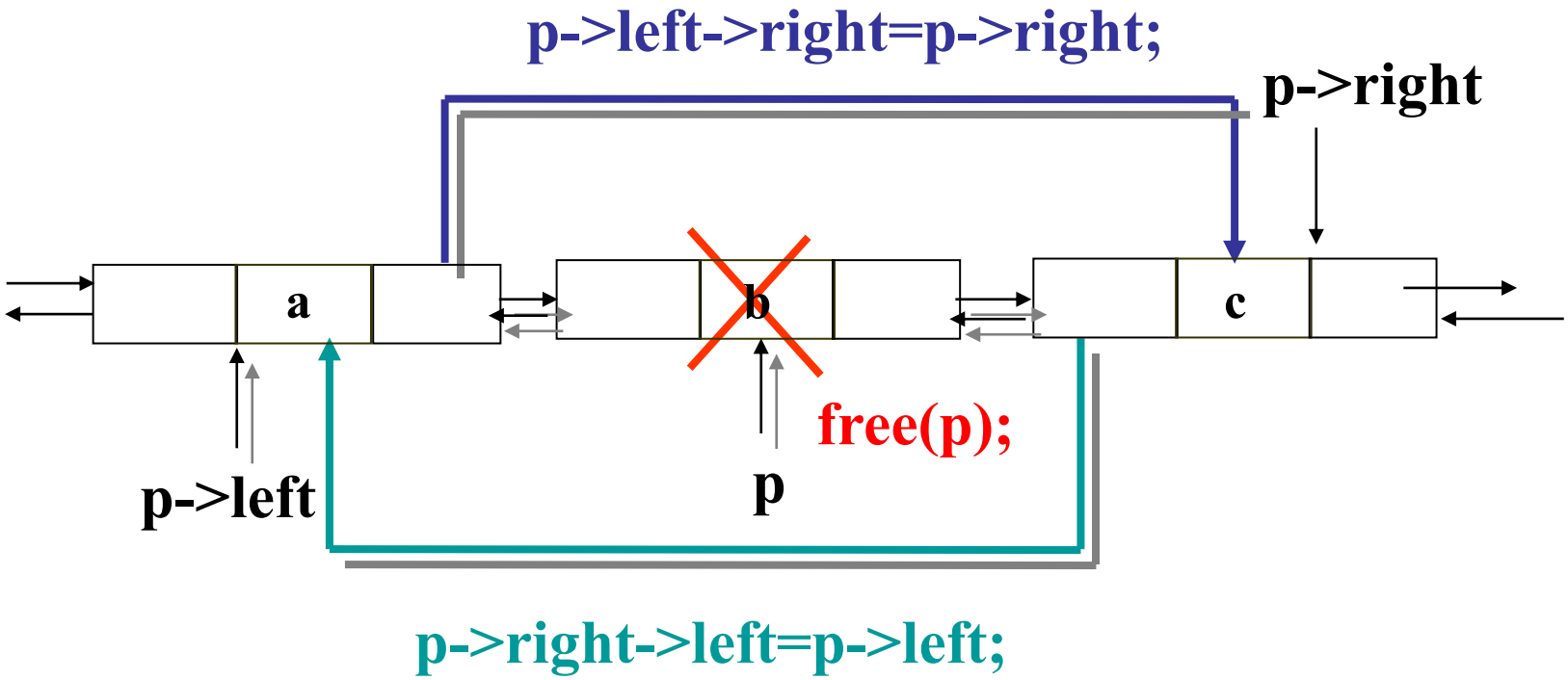
- 动因：从双链到双循环链类似于从单链到单循环链。
- 构想：引入哨兵结点，并将双链表的首、尾结点的空指针改为指向哨兵结点的指针，而哨兵结点的左、右指针分别指向尾元素、首元素所在的结点，使整个链表形成一个双链环。然后，用指向哨兵结点的指针表征我们的**ADT**表。结构如下图。



- 插入过程：


p->right
p
a
b
p->right = y;
p->right->left = y;
x
y->left = p;
y->right = p->right;
y
问题：以上语句顺序能否对调？

- 删除过程：


p->left->right=p->right;
p->right
a
b
c
free(p);
p->left
p
p->right->left=p->left;

3、用双循环链实现表的优缺点

- 优点:(1)可在O(1)的时间里找到表中任意一个元素的前驱;
  (2)可简化一些基本运算。
- 缺点：以更多的存储空间为代价

返回章目录

2.8 表的搜索游标

1、表的搜索游标的作用：

在对表进行各种操作时，常需要对表进行顺序扫描。为了使这种顺序扫描具有通用性，可将与之相关的运算定义为ADT表的基本运算。

2、表的搜索游标常用的基本运算：

(1) IterInit(L)：初始化搜索游标

(2) IterNext(L)：当前搜索游标的下一个位置

(3) CurrItem(L)：当前搜索游标处的表元素

(4) InsertCurr(x,L)：在当前搜索游标处插入元素x

(5) DeleteCurr(L)：删除当前搜索游标处的元素

返回章目录

2.9 表的应用举例

——Josephus排列问题

★问题描述：

Josephus 排列问题定义如下：假设 n 个竞赛者排成一个环形。给定一个正整数 m，从某个指定的第 1 个人开始，沿环计数，每遇到第 m 个人就让其出列，且计数继续进行下去。这个过程一直进行到所有的人都出列为止。最后出列者为优胜者。每个人出列的次序定义了整数 1，2，…，n 的一个排列。这个排列称为一个(n, m) Josephus 排列。

例如，(7, 3) Josephus 排列为 3，6，2，7，5，1，4。

★编程任务：

(1) 用抽象数据类型表设计一个求(n, m) Josephus 排列的算法。

(2) 试设计一个算法，对于给定的正整数 n 和 k，1≤k≤n，求正整数 m，使(n, m) Josephus 排列的最后一个数是 k。

返回章目录

THE END