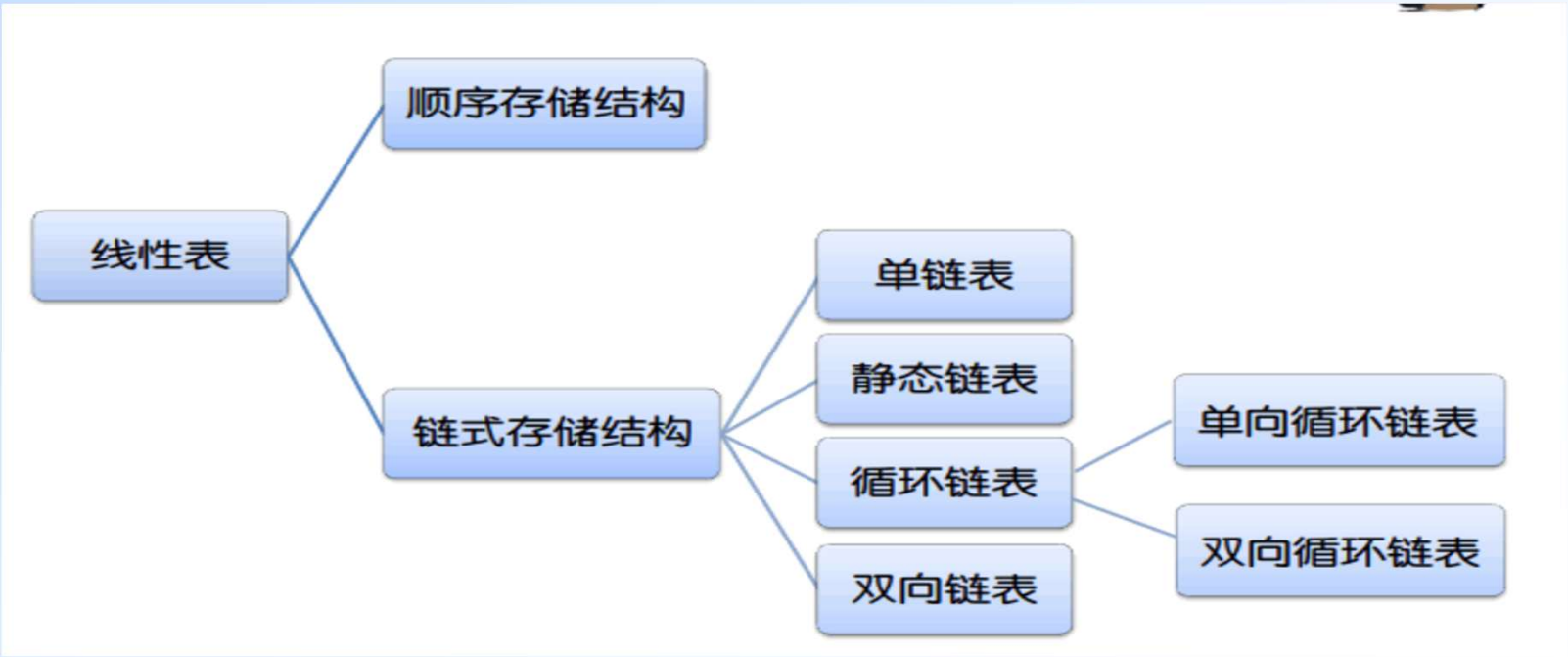
第二章 线性表

- 线性表
- 顺序表
- 链表
- 顺序表与链表的比较

线性表的学习内容

- 线性表是最简单、最常用的数据结构
- 顺序存储、链式存储是最常用的存储方法



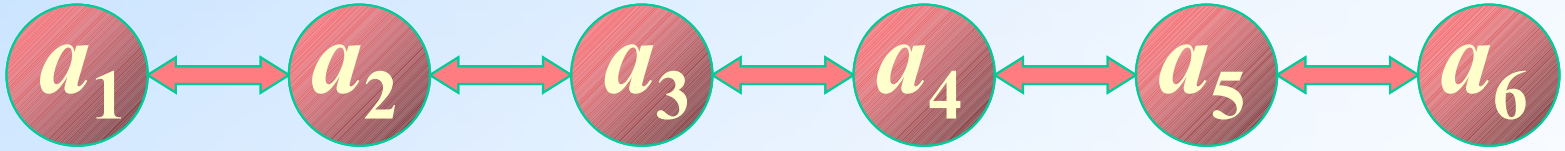
线性表

定义： n（≥0）个数据元素的有限序列，记作

$(a_1, \cdots a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$

其中，a_i是表中数据元素

n 是表长度，n=0时为空表



数据元素的具体含义在不同情况下各不相同

在复杂数据表中，一个数据元素由若干数据项组成

线性表

定义： n（≥0）个数据元素的有限序列，记作

$$(a_1, \cdots a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

特点:

- 同一线性表中元素具有相同特性
- 相邻数据元素之间存在序偶关系（线性表通过结点之间的位置确定结点之间的相互关系）
- 除第一个元素外，其他每一个元素有一个且仅有一个直接前驱。
- 除最后一个元素外，其他每一个元素有一个且仅有一个直接后继。

ADT

ADT List{

数据对象： D = { ai | 1<= i <= n,n>=0,ai 属于ElemType类型}

数据关系： R={<ai, ai+1> | ai, ai+1属于D, i=1...n-1}

基本运算：

initList(&L): 初始化线性表

destroyList(&L): 销毁线性表

listEmpty(L): 线性表是为空

listLength(L): 线性表的长度

dispList(L): 输出线性表

getElem(L,i,&e):求线性表某元素

locateElem(L,e):按元素查找

listInsert(&L, i, e):插入元素

listDelete(&L,i,e):删除元素

}ADT List

//详细定义参考书上ADT List

ADT

两个线性表，LA LB分别表示两个集合A和B，求新的集合A=AUB。

```
void union（List &La，List Lb）{
// 将所有在线性表Lb中但不在La中的数据元素插入到La中
La_len=ListLength（La）；
Lb_len=ListLength（Lb）；
//求线性表的长度

for（i=1；i<=Lb_len;i++）{
    GetElem(Lb,i,e); //取Lb中第i个数据元素赋给e
    if （！LocateElem（La，e，equal））
        ListInsert（La，++La_len，e）
    // La中不存在和e相同的数据元素，则插入
    }
}// union
```

ADT

两个线性表，LA LB分别表示两个集合A和B，求新的集合A=AUB。

```
void union（List &La，List Lb）{
// 将所有在线性表Lb中但不在La中的数据元素插入到La中
La_len=ListLength（La）；
Lb_len=ListLength（Lb）；
//求线性表的长度

for（i=1；i<=Lb_len;i++）{
     GetElem(Lb,i,e); //取Lb中第i个数据元素赋给e
    if （！ LocateElem（La，e，equal））
            ListInsert（La，++La_len，e）
     // La中不存在和e相同的数据元素，则插入
     }
}// union
```

对于union操作，用到了线性表基本操作ListLength、GetElem、LocateElem、Listlnsert等，对于复杂的个性化的操作，把基本操作组合起来实现的

顺序表

定义：将线性表中的元素相继存放在一个连续的存储空间中。

存储结构：数组

特点：线性表的顺序存储方式

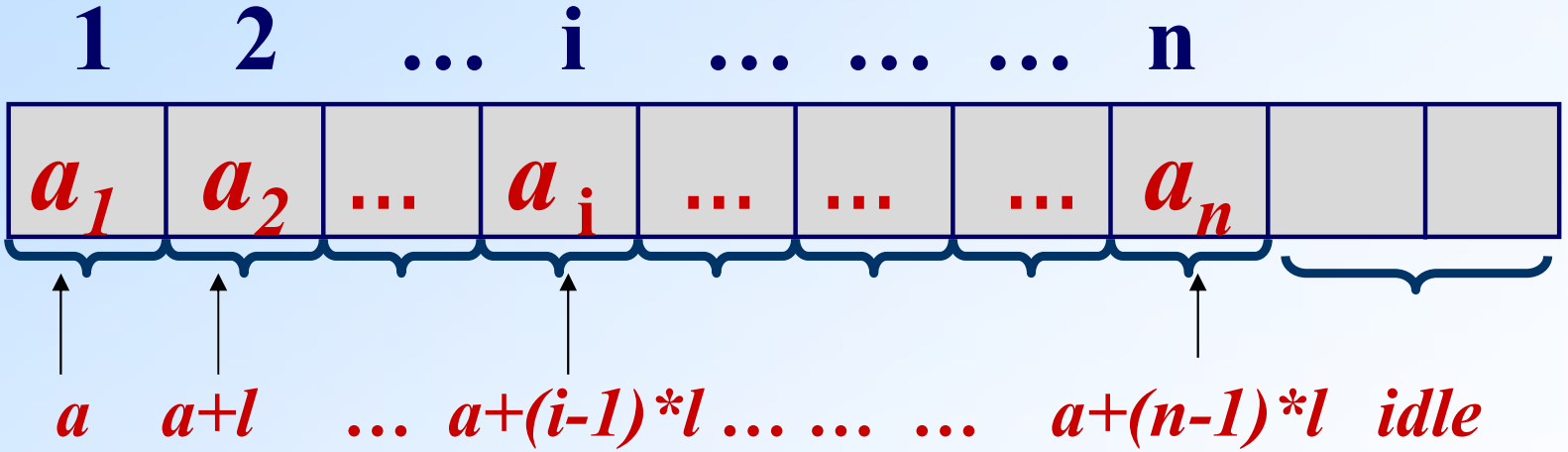
存取方式：顺序存取、随机存取

顺序存储结构示意图

顺序表的存储方式：

$LOC(a_{i+1}) = LOC(a_i)+1$

$LOC(a_i) = LOC(a_1)+(i-1)*l$



顺序表（SeqList）的类型定义

```
#define ListSize 100      //最大允许长度
typedef int ListData;

typedef struct {
   ListData * data; //存储空间基址
   int length;          //当前元素个数
} SeqList;
```

顺序表基本运算

- 初始化,空表长度为0

```
void InitList ( SeqList & L ) {
    // L.data = new ListData[ListSize];
    L.data = ( ListData * ) malloc
        ( ListSize * sizeof ( ListData ) );
    if ( L.data == NULL ) {
        printf ( "存储分配失败!\n" ); //cout<< ".." ;
        exit (1);
    }
    L.length = 0;
}
```

- SeqList & L 为什么引用传递?
  还可以如何传参数?
- 可以使用L= new ListData[ListSize]

- 求表的长度

```
int Length ( SeqList & L ) {
    return L.length;
}
```

- 提取函数：在表中提取第 i 个元素的值

```
ListData GetData ( SeqList &L, int i ) {
    if ( i >= 0 && i < L.length )
        return L.data[i];
    else printf ( “参数 i 不合理！\n” );
}
//按位置查找
```

可以用[]进行数组数据元素访问吗?

L.data[i]

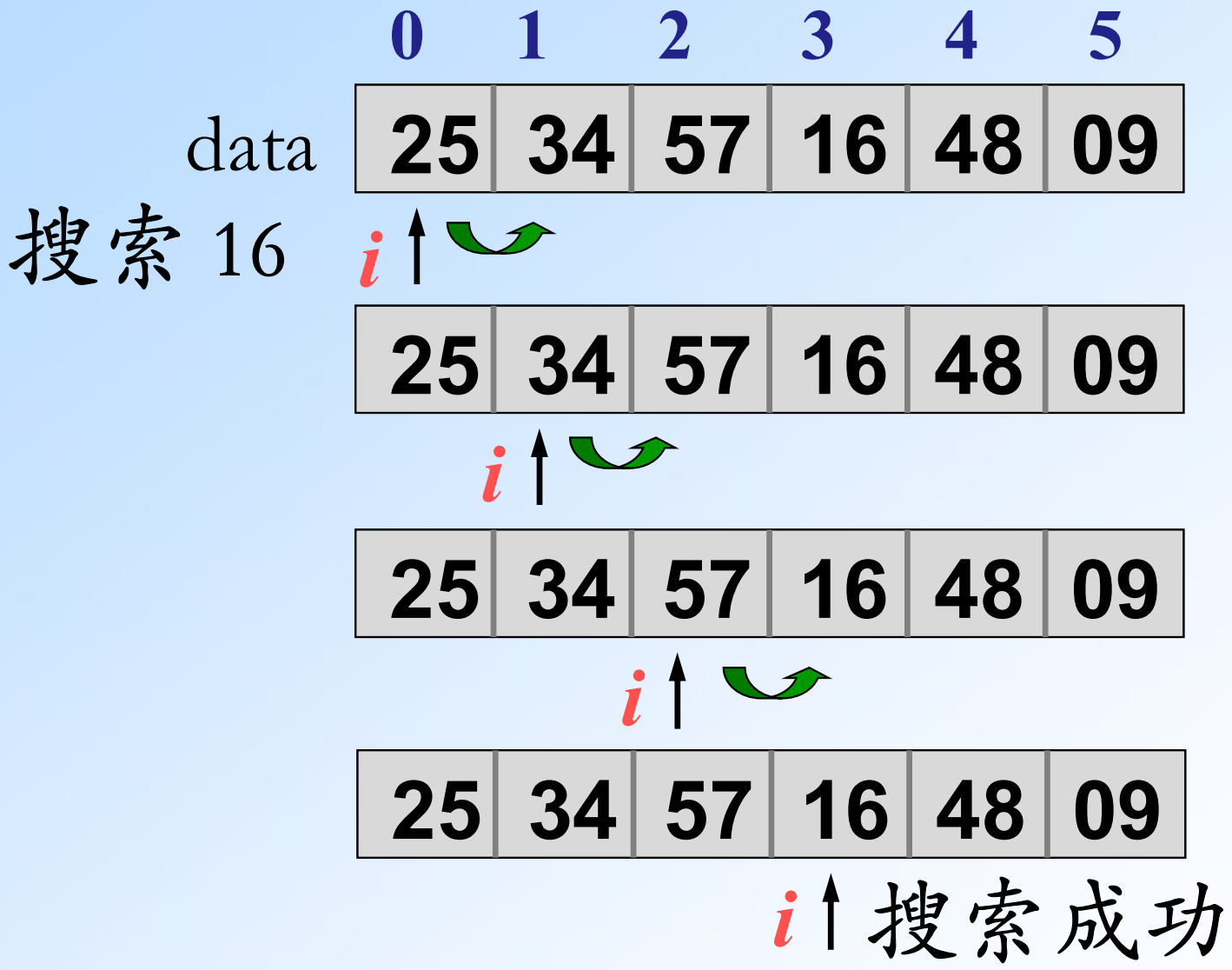
L[i]

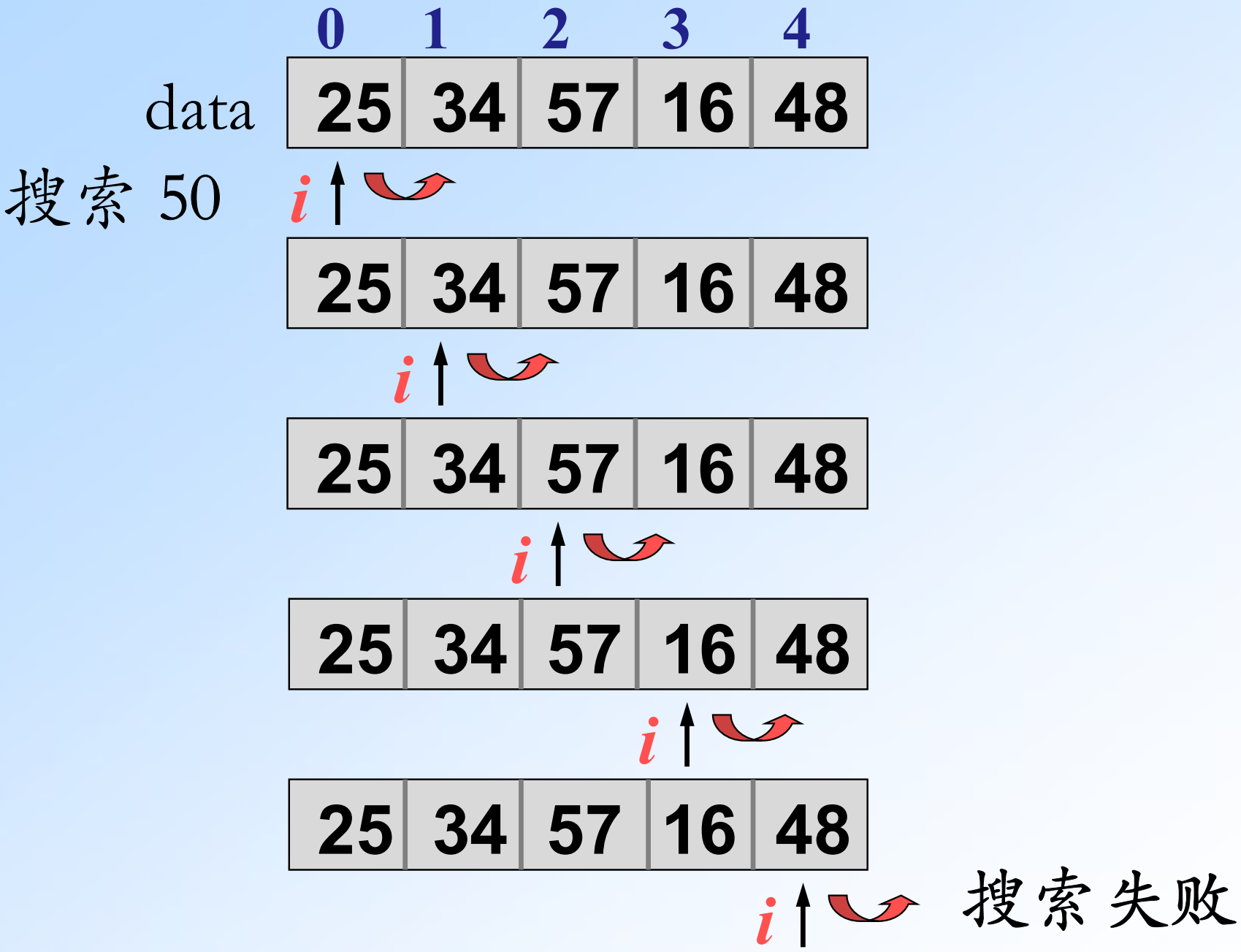
- 按值查找：找x在表中的位置，若查找成功，返回表项的位置，否则返回-1

```
int Find ( SeqList &L, ListData x ) {
    int i = 0;
    while ( i < L.length && L.data[i] != x )
        i++;
    if ( i < L.length ) return i;
    else return -1;
}
```

注意i < L.length && L.data[i] != x 的判断顺序

顺序搜索图示


0 1 2 3 4 5
data
25 34 57 16 48 09
搜索 16
i
25 34 57 16 48 09
i
25 34 57 16 48 09
i
25 34 57 16 48 09
i 搜索成功


0 1 2 3 4
data
25 34 57 16 48
搜索 50
i
25 34 57 16 48
i
25 34 57 16 48
i
25 34 57 16 48
i
25 34 57 16 48
i
搜索失败

搜索成功的平均比较次数

$$\mathrm{ACN} = \sum_{i=0}^{n-1} p_i \times c_i$$

若搜索概率相等，则

$$\mathrm{ACN} = \frac{1}{n}\sum_{i=0}^{n-1}(i+1) = \frac{1}{n}(1+2+\cdots+n) =$$

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

搜索不成功　数据比较 n 次

- 按值查找：判断x是否在表中

```
int IsIn ( SeqList &L, ListData x )
{
        int i = 0,  found=0;
        while ( i < L.length &&!found )
                if(L.data[i] != x ) i++;
                else found=1;
       return found;
}
```

- 按值查找：寻找x的后继

```
int Next ( SeqList &L, ListData x ) {
    int i = Find(x);
    if ( i >=0 && i < L.length-1 ) return i+1;
      else return -1;
}
```

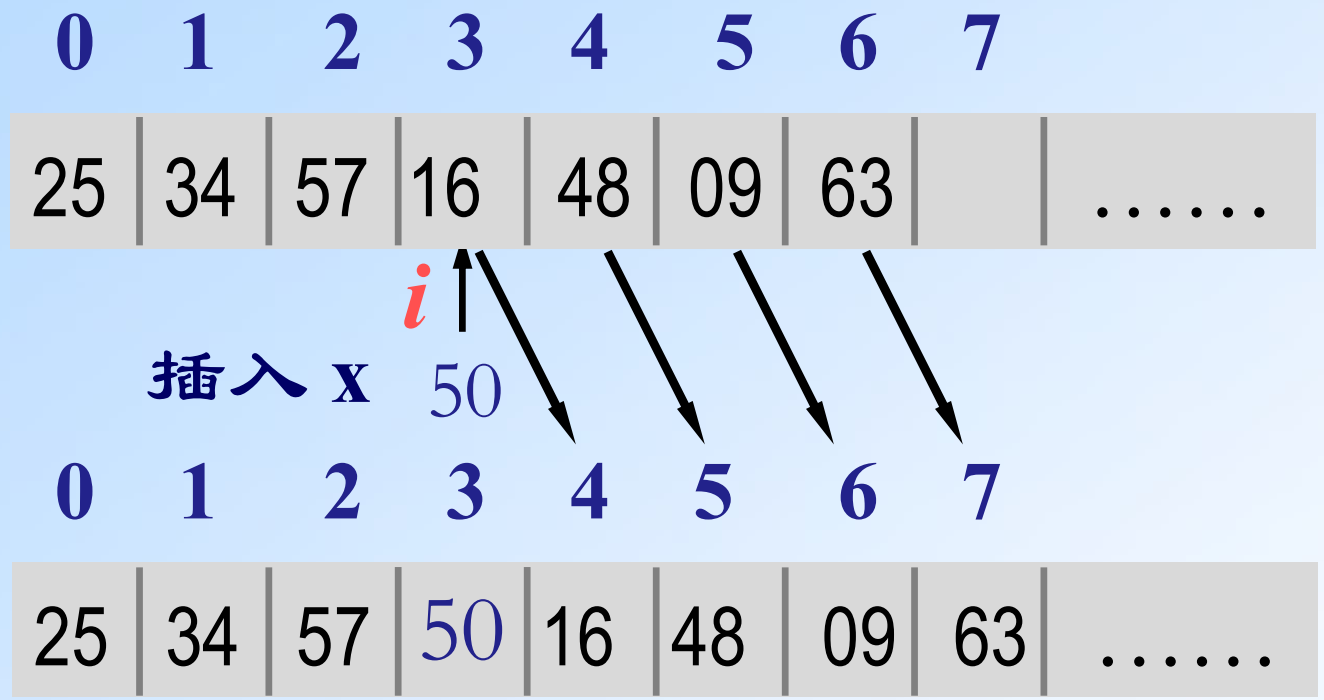
- 按值查找：寻找x的前驱

```
int Previous ( SeqList &L, ListData x ) {
    int i = Find(x);
    if ( i >0 && i < L.length ) return i-1;
      else return -1;
}
```

- 顺序表的插入

```
//在表中第 i 个位置插入新元素 x
int Insert ( SeqList &L, ListData x, int i ) {
if (i < 0 || i > L.length || L.length == ListSize)
    return 0;            //插入不成功
else {
      for ( int j = L.length; j > i; j-- )
            L.data[j] = L.data[j -1];
      L.data[i] = x;
      L.length++;
      return 1;     //插入成功
  }
}
```

插入



顺序表插入时，平均数据移动次数AMN在各表项插入概率相等时

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

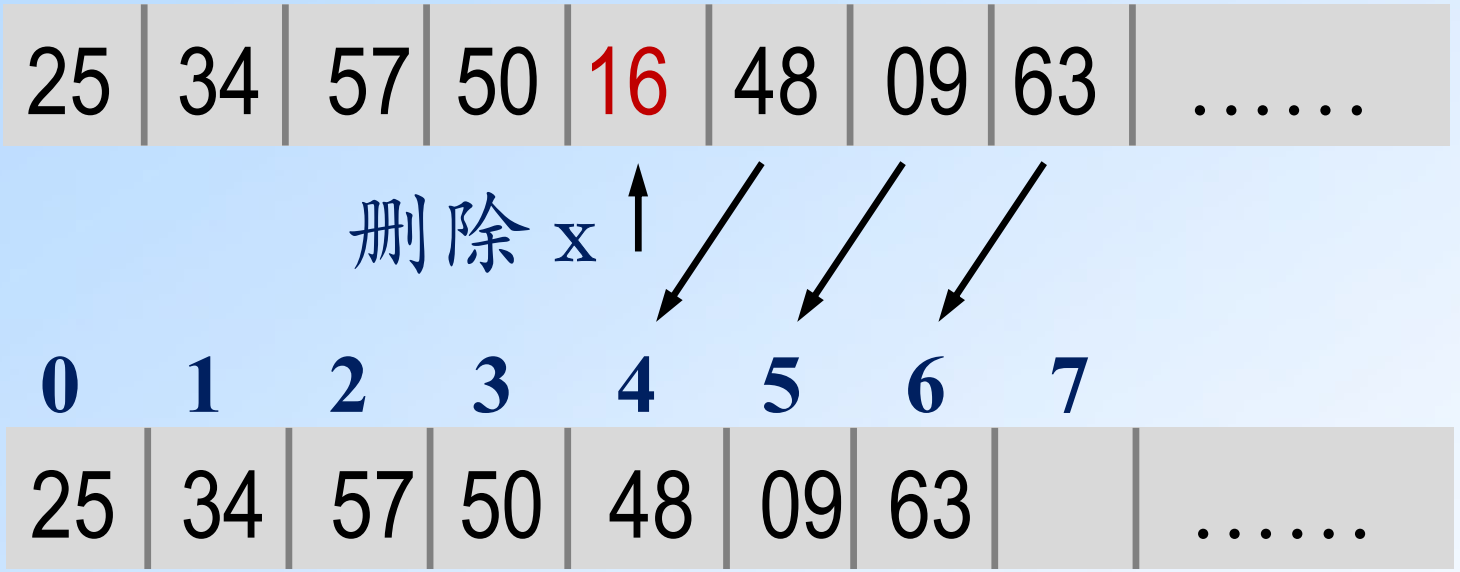
$$= \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

n为表内数据元素个数

- 顺序表的删除

```
//在表中删除已有元素 x
int Delete ( SeqList &L, ListData x ) {
    int i = Find (L, x); //在表中查找 x
    if ( i >= 0 ) {
        L.length -- ;
        for ( int j = i; j < L.length; j++ )
                L.data[j] = L.data[j+1];
        return 1; //成功删除
    }
   return 0; //表中没有 x
}
```

删除



顺序表删除平均数据移动次数AMN在各表项删除概率相等时

$$AMN = \frac{1}{n}\sum_{i=0}^{n-1}(n-i-1) = \frac{1}{n}\frac{(n-1)n}{2} = \frac{n-1}{2}$$

n为表内数据元素个数

顺序表的应用:集合的“并”运算

```
void Union ( SeqList &A, SeqList &B ) {
    int n = Length ( A );
    int m = Length ( B );
    for ( int i = 0; i < m; i++ ) {
        int x = GetData ( B, i );  //在B中取一元素
        int k = Find (A, x);       //在A中查找它
        if ( k == -1 )             //若未找到插入它
          { Insert ( A, x, n );  n++; }
    }
}
```

执行后A，B改变了吗?

如果C=A并B，程序如何改?

顺序表的应用:集合的“交”运算

```
void Intersection ( SeqList &A, SeqList &B ) {
    int n = Length ( A );
    int m = Length ( B );
    int i = 0;
    while ( i < n ) {
        int x = GetData ( A, i );  //在A中取一元素
        int k = Find ( B, x );    //在B中查找它
        if ( k == -1 ) { Delete ( A, i );  n--; }
        else i++;                  //未找到在A中删除它
    }
}
```

执行后A，B改变了吗?

如果C=A交B，程序如何改?

下面是采用C++描述的顺序表的定义
及操作，供实验参考

顺序表(SeqList)类的定义

```
class SeqList {
    int *data;                  //顺序表存储数组
    int MaxSize;                //最大允许长度
    int last;                   //当前最后元素下标
public:                         构造函数，复制构造函数
    SeqList ( int MaxSize = defaultSize );
    ~SeqList ( ) { delete [ ] data; }
    int Length ( ) const { return last+1; }
```

```
    int Find ( int& x ) const;          //查找
    int Locate ( int i ) const;         //定位
    int Insert ( int & x, int i );      //插入
    int Remove ( int & x );             //删除
    int Next ( int & x ) ;              //后继
    int Prior ( int & x ) ;             //前驱
    int IsEmpty ( ) { return last ==-1; }
    int IsFull ( ) { return last == MaxSize-1; }
    int Get ( int i ) {                 //提取
        return i < 0 || i > last?  NULL : data[i];
    }
                          output（），changeLength（）
}
```

顺序表部分公共操作的实现

```
//构造函数
SeqList:: SeqList ( int sz ) {
    if ( sz > 0 ) {
        MaxSize = sz;  last = -1;
        data = new int[MaxSize];
        if ( data == NULL ) {      //空间申请失败
            MaxSize = 0;   last = -1;
            return;
        }
    }
}
```

```
int SeqList:: Find ( int & x ) const {
//搜索函数：在顺序表中从头查找结点值等于
//给定值x的结点
   int  i = 0;
      while ( i <= last && data[i] != x )
          i++;
      if ( i > last ) return -1;
      else return i;
}
```

```
int SeqList:: Insert (int& x, int i ) {
//在表中第 i 个位置插入新元素 x
    if ( i < 0 || i > last+1 || last == MaxSize-1 )
          return 0;                  //插入不成功
    else {
        last++;

         for ( int j = last; j > i; j-- )
              data[j] = data[j -1];
        data[i] = x;   return 1;    //插入成功
    }
}
```

```
int SeqList:: Remove ( int& x ) {
 //在表中删除已有元素 x
    int i = Find (x);        //在表中搜索 x
    if ( i >= 0 ) {
        last-- ;
         for ( int j = i; j <= last; j++ )
            data[j] = data[j+1];
        return 1;                    //成功删除
    }
    return 0;                        //表中没有 x
 }
```

顺序表的应用：集合的“并”运算

```
void Union ( SeqList& A, SeqList& B ) {
    int n = A.Length ( );
    int m = B.Length ( );
    for ( int i = 0; i < m; i++ ) {
        int x = B.Get(i);          //在B中取一元素
        int k = A.Find (x);        //在A中搜索它
        if ( k == -1 )             //若未找到插入它
          { A.Insert (n, x);  n++; }
    }
}
```

成员函数的参数是什么？

顺序表的应用：集合的“交”运算

```
void Intersection ( SeqList& A, SeqList& B ) {
    int n = A.Length ( );
    int m = B.Length ( );
    int i = 0;
    while ( i < n ) {
        int x = A.Get (i);      //在A中取一元素
        int k = B.Find (x);     //在B中搜索它
        if ( k == -1 ) { A.Remove (i);  n--; }
        else i++;               //未找到在A中删除它
    }
}
```

课堂练习

- 设计算法，将顺序表的所有元素逆置

课堂练习

- 设计算法，将顺序表的所有元素逆置

```
void Reverse(SeqList &L){
    ElemType temp;
    for(i = 0; i < L.length/2; i++){
        temp = L.data[i];
        L.data[i] = L.data[L.length-i-1];
        L.data[L.length-i-1] = temp;
    }
}
```

顺序表的优缺点

- 优点:

简单、运算方便，特别是对于小线性表或长度固定的线性表，采用顺序存储结构的优越性更为突出；

随机访问效率高

- 缺点:

顺序存储插入与删除一个元素，需要移动若干个数据元素，对大的线性表，特别元素的插入和删除频繁时，顺序存储效率低；（如果不要求按大小排序存储，有优化方法吗？）

顺序存储空间分多了浪费，分少了空间不足上溢，存储空间不便扩充

链表（Linked List）

链表是线性表的链接存储表示

- 单链表
- 静态链表
- 循环链表
- 双向链表

单链表

定义： 用一组地址任意的存储单元存放线性表中的数据元素。

头指针

31

表中节点位置	存储地址	数据域	指针域
6	1	ZHANG	13
5	7	WANG	1
7	13	LI	null
2	19	ZHAO	37
4	25	WU	7
1	31	ZHOU	19
3	37	SUN	25

存储线性表 ZHOU,ZHAO,SUN,WU,WANG,ZHANG,LI

单链表结构

每个元素由结点(Node)构成，它包括两个域:数据域Data和指针域Link

存储结构： 链式存储结构

特点： 存储单元可以不连续

存取方式： 顺序存取

单链表的类型定义

是不是可以这样定义?

```
typedef struct node
    ListData  data;
    ListData  * link
} ListNode
```

```
typedef char ListData;

typedef struct node {          //链表结点
    ListData  data;                 //结点数据域
    struct node  * link;          //结点链域
} ListNode;

typedef ListNode  * LinkList;   //链表头指针
LinkList  first;                    //链表头指针
```

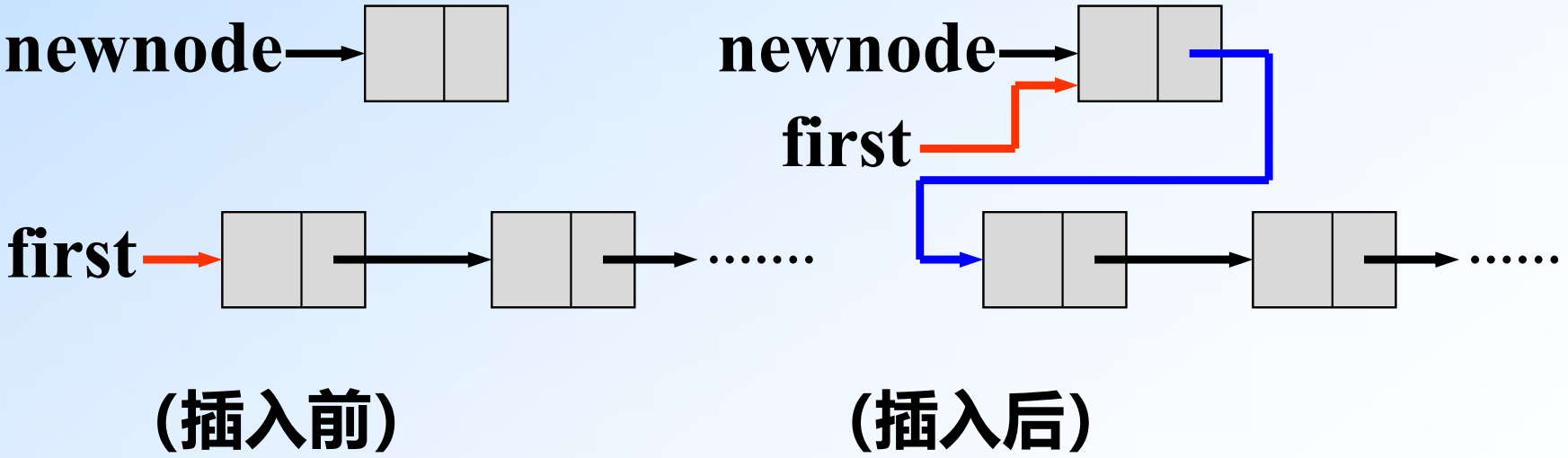
C++如何定义?

单链表的基本运算

- 插入（三种情况）

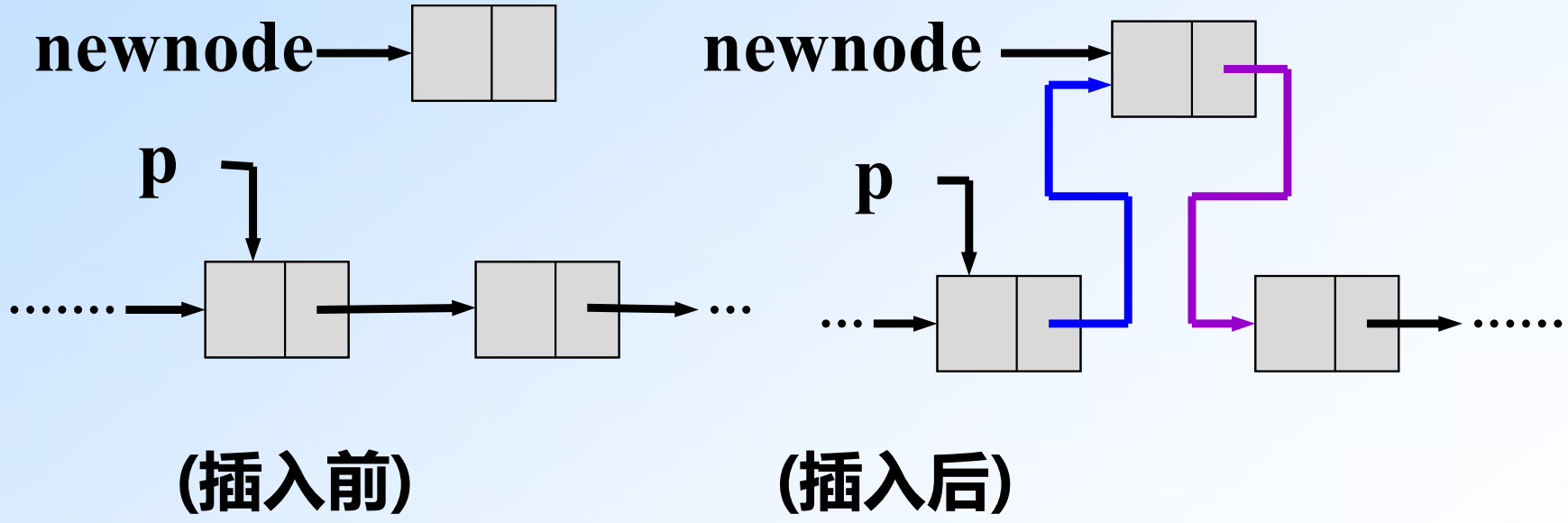
第一种情况：在第一个结点前插入

```
newnode->link = first ;
first = newnode;
```



第二种情况：在链表中间插入

```
newnode->link = p->link;
p->link = newnode;
```

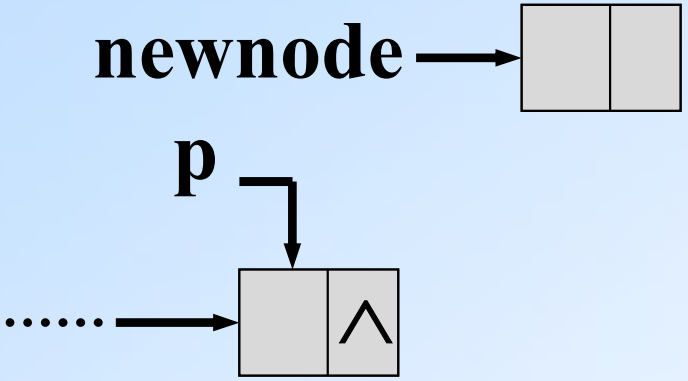


两条语句可以交换吗?

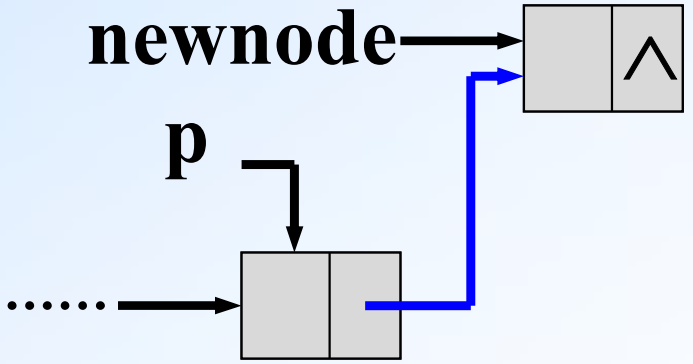
第三种情况：在链表末尾插入

newnode->link = null
p->link = newnode;

newnode->link = p->link;
p->link = newnode;



(插入前)



(插入后)

第一条语句中p->link是什么？

三种插入代码

```
newnode->link = first ;
first = newnode;
```

```
newnode->link = p->link;
p->link = newnode;
```

```
newnode->link = p->link;
p->link = newnode;
```

为什么不一样?
P是哪个结点?

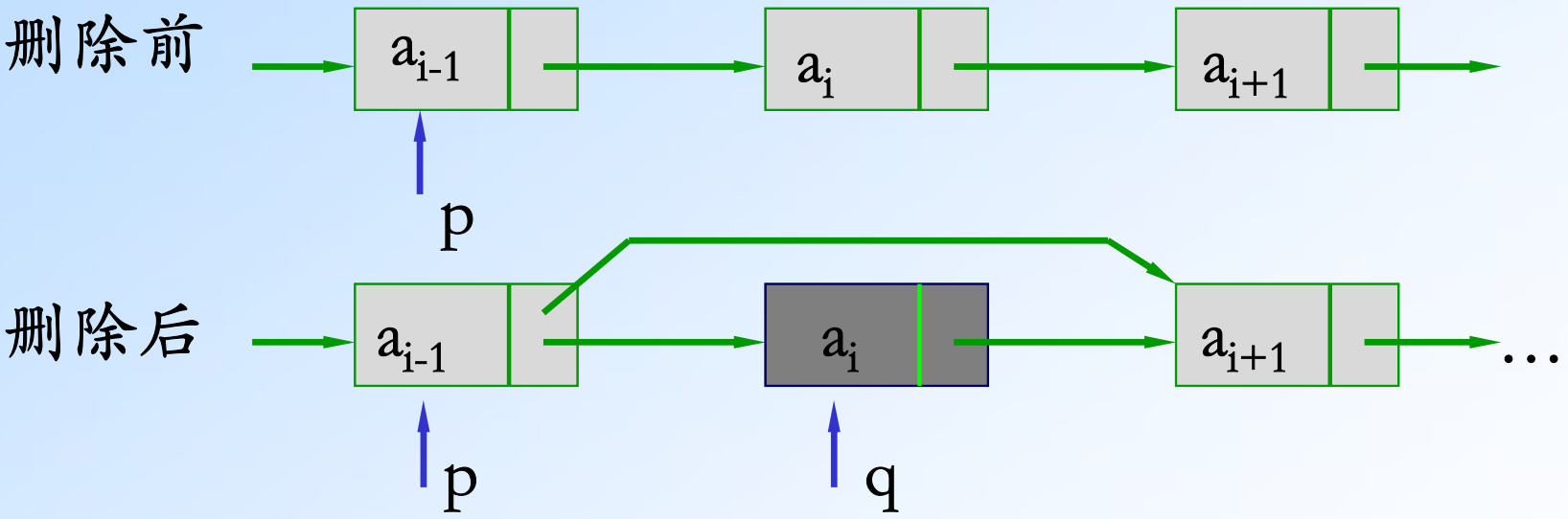
```
int ListInsert (int x, int i ) {
//在链表第 i 个结点处插入新元素 x
   ListNode * p = first;
   for ( k = 0; k < i -1; k++ )  //找第 i-1个结点
     if ( p == NULL ) break;
     else p = p->link;
   if ( p == NULL && first != NULL ) {
      printf( “无效的插入位置!\n” );
      return 0;
   }
ListNode *newnode =          //创建新结点
       (ListNode*) malloc(sizeof (ListNode));
if ( first == NULL || i == 0 ) {  //插在表前
        newnode->link = first;
        if ( first == NULL ) last = newnode;
        first = newnode;
    }
    else {                        //插在表中或末尾
       newnode->link = p->link;
       if ( p->link == NULL ) last = newnode;
       p ->link = newnode;
    }
    return 1;
 }
```

```
int ListInsert (int x, int i ) {
//在链表第 i 个结点处插入新元素 x
    ListNode * p = first;
    for ( k = 0; k < i -1; k++ )  //找第 i-1个结点
      if ( p == NULL ) break;
      else p = p->link;
    if ( p == NULL && first != NULL ) {
       printf( “无效的插入位置!\n” );
       return 0;
    }
ListNode *newnode =              //创建新结点
        (ListNode*) malloc(sizeof (ListNode));
```

```
if ( first ==NULL || i ==0 ) {  //插在表前
        newnode->link = first;
        if ( first == NULL ) last = newnode;
        first = newnode;
    }
    else {                          //插在表中或末尾
        newnode->link = p->link;
        if ( p->link == NULL ) last = newnode;
        p ->link = newnode;
    }
    return 1;
}
```

- 删除： 在单链表中删除a_i结点

```
q = p->link;
p->link = q->link;
```



q的作用

p->link = p->link ->link;行不行呢?

```
int ListRemove ( int i ) {
   //在链表中删除第 i 个结点
   ListNode *p, *q;
   if ( i == 0 )          //删除表中第 1 个结点
     { q = first;  p = first = first->link; }
   else {
     p = first;              int k = 0;   //找第 i-1个结点
     while ( p != NULL && k < i-1 )
       { p = p->link; k++; }
     if ( p == NULL || p->link == NULL ) {
       printf( "无效的删除位置!\n" ); return 0;
     }
     else {          //删除表或表尾元素
      q = p->link;      //重新链接
      p->link = q->link;
     }
   if ( q == last ) last = p;                //可能修改last
   int x = q->data;
   free(q);                //删除q
   return x;               //返回第 i 个结点的值
}
```

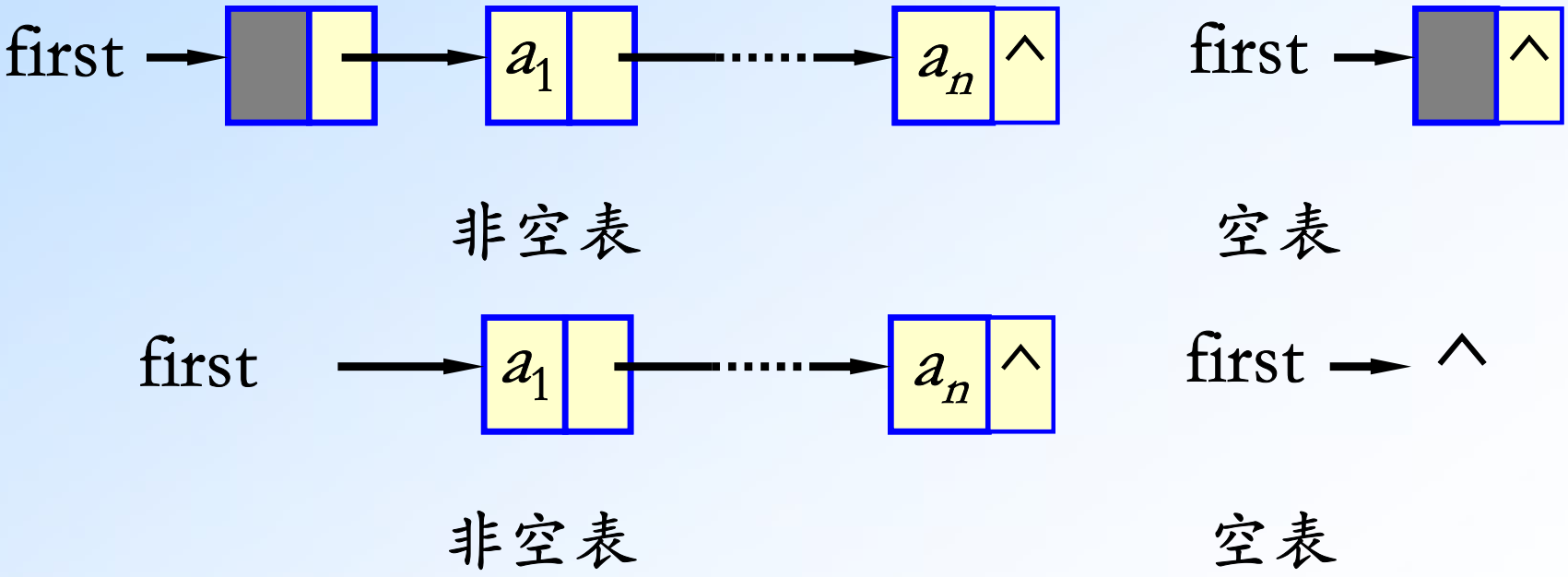
```
    else {          //删除表中或表尾元素
        q = p->link;      //重新链接
        p->link = q->link;
    }
    if ( q == last ) last = p;          //可能修改last
    int x = q->data;
    free(q);                  //删除q
    return x;       //返回第 i 个结点的值
}
```

p、q分别指向哪个结点?

带表头结点的单链表

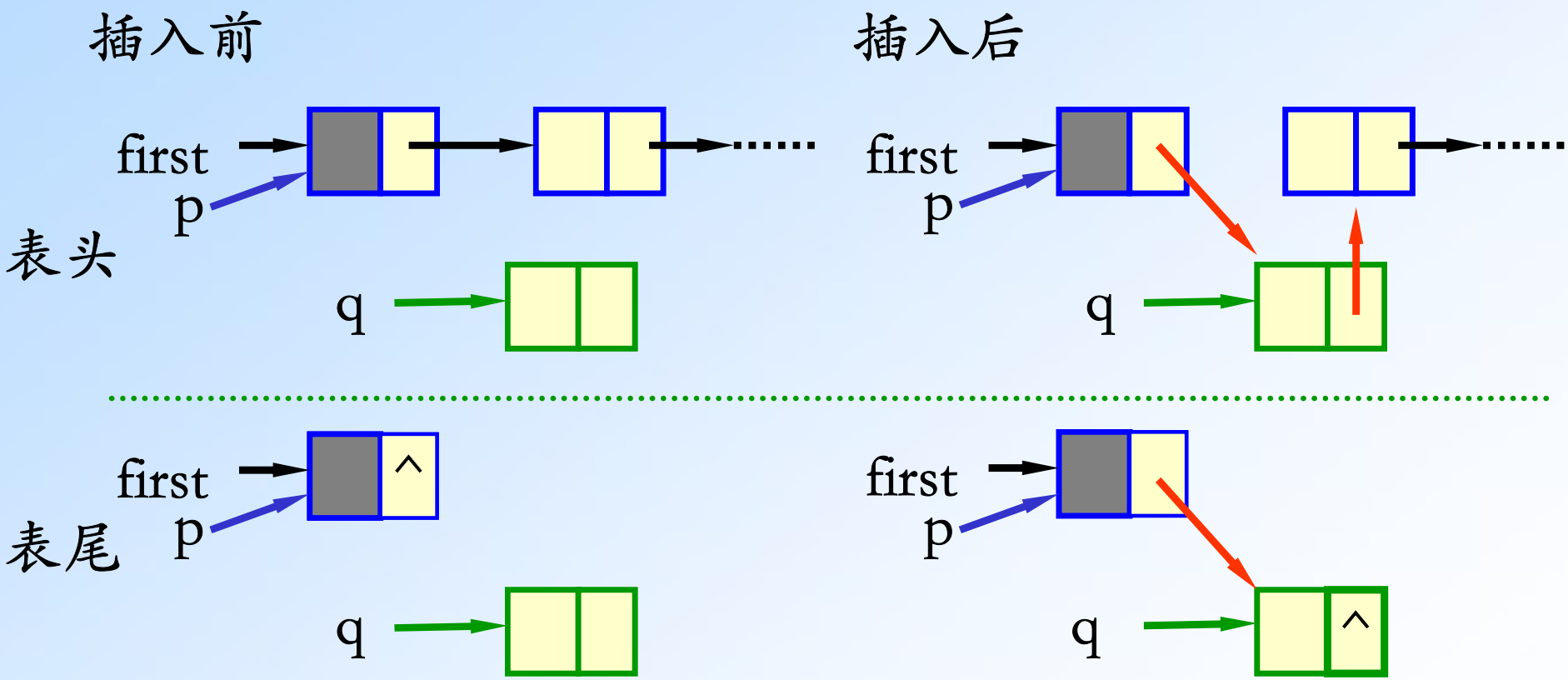
表头结点位于表的最前端，本身不带数据，仅标志表头。

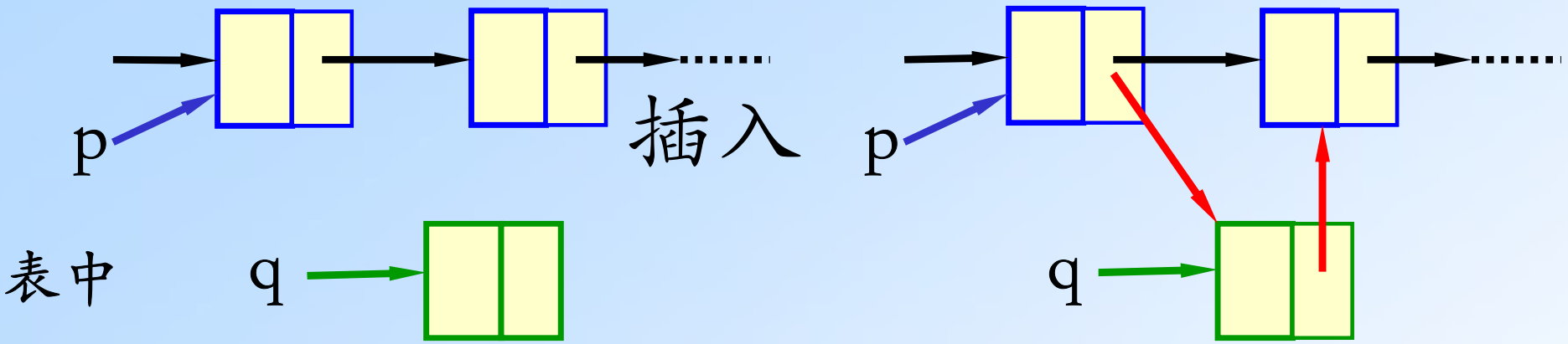
设置表头结点的目的:简化链表操作的实现



插入　　q->link = p->link;

p->link = q;



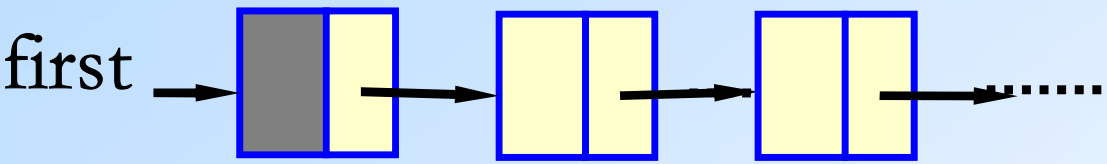

p
插入
表中
q
p
q

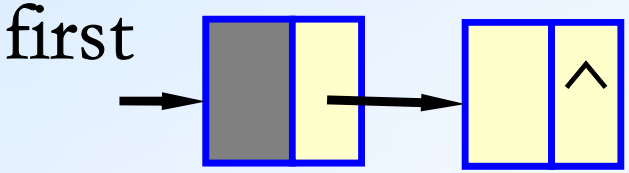
```
int Insert (LinkList first, ListData x, int i ) {
//将新元素 x 插入在链表中第 i 号结点位置
    ListNode * p = Locate ( first, i-1 );
    if ( p == NULL ) return 0;        //参数i值不合理返回0
    ListNode * newnode =              //创建新结点
        (ListNode *) malloc (sizeof (ListNode) );
    newnode->data = x;
    newnode->link = p->link;          //链入
    p->link = newnode;
    return 1;
}
```

删除

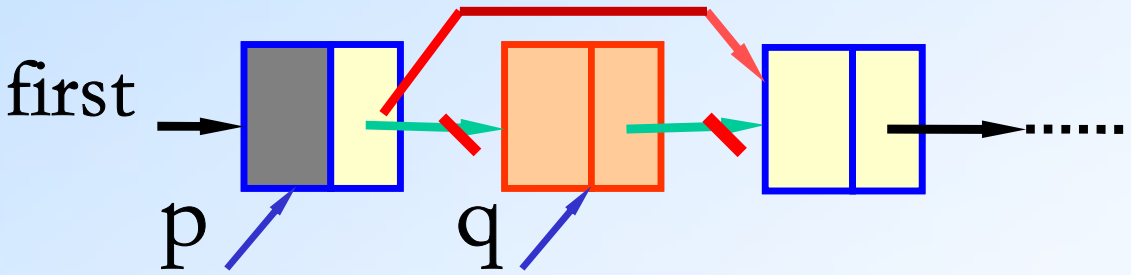
```
q = p->link;
p->link = q->link;
delete q;
```

删除前

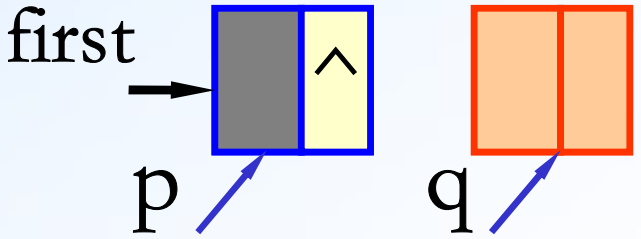




删除后



(非空表)

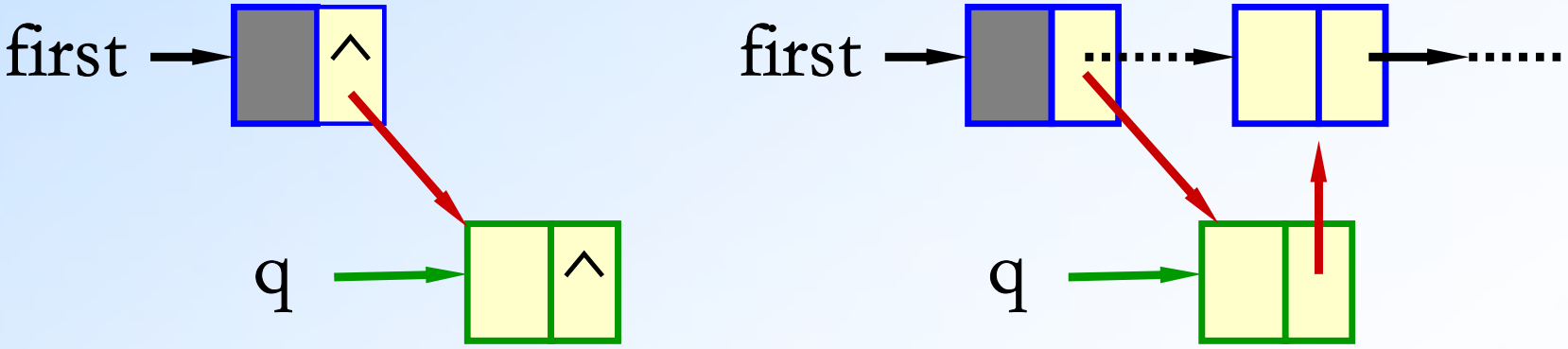


(空表)

```
int delete ( LinkList first, int i ) {
    //将链表第 i 号元素删去
    ListNode * p, * q
    p = Locate ( first, i-1 ); //寻找第i-1个结点
    if ( p == NULL || p->link == NULL)
        return 0;//i值不合理或空表
    q = p->link;
    p->link = q->link;              //删除结点
    free ( q );                     //释放
    return 1;
}
```

前插法建立单链表

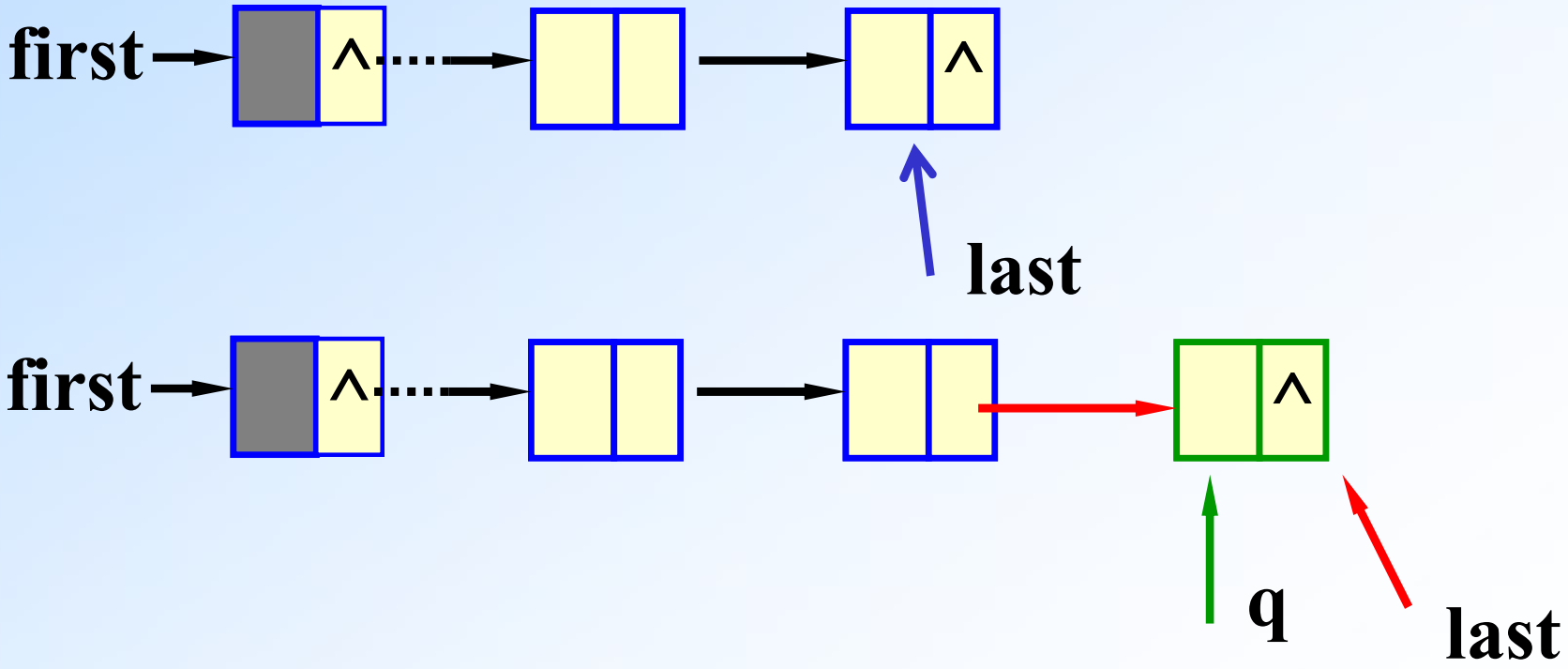
- 从一个空表开始，重复读入数据：
  - 生成新结点
  - 将读入数据存放到新结点的数据域中
  - 将该新结点插入到链表的前端
  - 直到读入结束符为止



```
LinkList createListF ( void ) {
    char ch;   ListNode *q;
    LinkList  head =                    //建立表头结点
        (LinkList) malloc (sizeof (ListNode));
    head->link = NULL;
    while ( (ch = getchar( ) ) !=  '\n'  ) {
        q = (ListNode *) malloc (sizeof(ListNode));
        q->data = ch;                   //建立新结点
        q->link = head->link;  //插入到表前端
        head->link = q;
    }
    return head;
}
```

后插法建立单链表

- 每次将新结点加在链表的表尾;
- 尾指针r,总是指向表中最后一个结点，新结点插在它的后面;



```
LinkList createListR ( void ) {
    char ch;
    LinkList  head =        //建立表头结点
        (LinkList) malloc (sizeof (ListNode));
    ListNode *q, *tail = head;
    while ( (ch = getchar( ) ) !=  '\n'  ) {
        q = (ListNode *) malloc (sizeof(ListNode));
         q->data = ch;            //建立新结点
         tail ->link = q;   tail =q;    //插入到表末端
    }
    tail ->link = NULL;
    return head;
}
```

单链表清空

```
void makeEmpty ( LinkList first ) {
//删去链表中除表头结点外的所有其它结点
    ListNode *q;
    //当链不空时，循环逐个删去所有结点
    while ( first->link != NULL ) {
        q = first->link;  first->link = q->link;
        free( q );        //释放
    }
    last=first;
}
```

- First=NULL是不是更简单?
- 前删还是后删?
- 为什么不从另一个方向删?

计算单链表长度

```
int Length ( LinkList first ) {
   ListNode *p = first->link; //指针 p 指示第一个结点
    int count = 0;
    while ( p != NULL ) {      //逐个结点检测
         p = p->link;  count++;
     }
     return count;
}
```

按值查找

```
ListNode * Find ( LinkList first, ListData value )
{
    //在链表中从头搜索其数据值为value的结点
     ListNode * p = first->link;
    //指针 p 指示第一个结点
     while ( p != NULL && p->data != value )
         p = p->link;
     return p;
}
```

- 如果没找到，返回什么？
- 可以使用while (p->data != value &&p != NULL)吗？

按序号查找（定位）

```
ListNode * Locate ( LinkList first, int i ) {
//返回表中第 i 个元素的地址
    if ( i < 0 ) return NULL;
    ListNode * p = first;
    int k = 0;
    while ( p != NULL && k < i )
       { p = p->link;  k++; }      //找第 i 个结点
    if  ( k == i ) return p; //返回第 i 个结点地址
    else return NULL;
}
```

顺序表&链表
基本操作时间复杂度

- 顺序表
  - 值查找
  - 位置查找
  - 插入
  - 删除
  - 前驱后继
- 链表
  - 值查找
  - 位置查找
  - 插入
  - 删除
  - 前驱后继

顺序表&链表
基本操作时间复杂度

- 顺序表
  - 值查找 O(n)----（n+1）/2
  - 位置查找 O(1)
  - 插入 O(n)---- n/2
  - 删除 O(n)----（n-1）/2
  - 前驱后继 O(1)
- 链表
  - 值查找 O(n)----（n+1）/2
  - 位置查找 O(n)----（n+1）/2
  - 插入 O(1)
  - 删除 O(1)
  - 前驱后继 O(n) O(1)

课堂练习：获取倒数第N个结点值

课堂练习：获取倒数第N个结点值

思路：建立两个指针，第一个先走n步，第2个指针开始走，当第一个结点走到链表末尾时，第二个节点的位置就是倒数第n个节点的值。

课堂练习：获取倒数第N个结点值

```
while (i < n && firstNode->next != NULL) {
    //正数N个节点，firstNode指向正的第N个节点
    i++;
    firstNode = firstNode->next;
}
while (firstNode != NULL) {
    //查找倒数第N个元素
    secNode = secNode->next;
    firstNode = firstNode->next;
}
```

课堂练习：删除单链表中的重复元素

课堂练习：删除单链表中的重复元素

思路：建立两个工作指针p,q，p遍历全表。

p每到一个结点，q从这个结点往后遍历，并与p的数值比较，相同的话删除掉那个结点

课堂练习：删除单链表中的重复元素

```
LinkList RemoveDupNode(LinkList L) {    //删除重复结点的算法
    LinkList p, q, r;
    p=L->next;
        while(p) {// p用于遍历链表
            q=p;
            while(q->next) {// q遍历p后面的结点，并与p数值比较
                if(q->next->data==p->data) {
                r=q->next; // r保存需要删掉的结点
                q->next=r->next; // 需要删掉的结点的前后结点相接
                free(r);
                }
                else q=q->next;
            }
         p=p->next;
        }
    return L;
    }
```

单链表的类定义(C++描述)

- 多个类表达一个概念(单链表)
  - 链表结点(ListNode)类
  - 链表(List)类
  - 链表游标(Iterator)类
- 定义方式
  - 复合方式
  - 嵌套方式
  - 继承方式

```
class List;                  //链表类定义（复合方式）

class ListNode {             //链表结点类
friend class List;           //链表类为其友元类
private:
    int data;                //结点数据
    ListNode * link;         //结点指针
};

class List {                 //链表类
private:
    ListNode *first , *last;     //表头指针
};
```

```
class List {          //链表类定义(嵌套方式)
private:
    class ListNode {          //嵌套链表结点类
    public:
        int data;
        ListNode *link;
    };
    ListNode *first , *last;    //表头指针
public:
    //链表操作……
};
```

链表类和链表结点类定义(继承方式)

```
class ListNode {          //链表结点类
protected:
    int data;
    ListNode * link;
};

class List : public class ListNode {
//链表类, 继承链表结点类的数据和操作
 private:
     ListNode *first , *last;       //表头指针
};
```

静态链表

用一维数组描述线性链表

0		1
1	ZHANG	2
2	WANG	3
3	LI	4
4	ZHAO	5
5	WU	-1
6		

修改前

0		1
1	ZHANG	2
2	WANG	6
3	LI	5
4	ZHAO	5
5	WU	-1
6	CHEN	3

(插入chen,删除zhao)修改后

静态链表和动态链表

- 相同点
  - 数据元素之间的逻辑关系依靠指针（游标）关联
- 存储空间
  - 静态链表存储数据元素的个数是预先确定的
  - 动态链表存储数据动态申请内存
- 空闲空间
  - 静态链表是在固定大小的存储空间内随机存储各个数据元素，需要使用另一条链表空闲空间位置，分配在新的元素
- 适用场景
  - 没有指针的语言也拥有链表结构
  - 小数据,操作频繁的场景

静态链表的定义

```
const int MaxSize = 100;      //静态链表大小
typedef  int ListData;

typedef struct node {         //静态链表结点
    ListData data;
    int link;
} SNode;

typedef struct {              //静态链表
    SNode Nodes[MaxSize];
    int newptr;               //当前可分配空间首地址
} SLinkList;
```

链表空间初始化

```
void InitList ( SLinkList* SL ) {
    SL->Nodes[0].link = -1;
    SL->newptr = 1;   //当前可分配空间从1 开始
    //建立带表头结点的空链表
    for ( int i = 1; i < MaxSize-1; i++ )
        SL->Nodes[i].link = i+1; //构成空闲链接表
     SL->Nodes[MaxSize-1].link = -1;
     //链表收尾
}
```

在静态链表中查找具有给定值的结点

```
int Find ( SLinkList SL, ListData x ) {
        //指针 p 指向链表第一个结点
        int p = SL.Nodes[0].link;
        while ( p != -1 )//逐个查找有给定值的结点
        if ( SL.Nodes[p].data != x)
            p = SL.Nodes[p].link;
        else break;
        return p;
}
```

p是什么？ --下标

在静态链表中查找第 i 个结点

```
int Locate ( SLinkList SL, int i ) {
    if ( i < 0 ) return -1;//参数不合理
    if ( i == 0 ) return 0;
    int j = 0, p = SL.Nodes[0].link;
    //循环查找第 i 号结点
    while ( p != -1 && j < i ) {
        p = SL.Nodes[p].link;
        j++;
    }
   return p;
}
```

在静态链表第 i 个结点处插入一个新结点

```
int Insert ( SLinkList* SL, int i, ListData x ) {
    int p = Locate ( SL, i-1 );
    if ( p == -1 ) return 0;            //找不到结点
    int q = SL->newptr;                 //分配结点
    SL->newptr = SL->Nodes[SL->newptr].link;
    SL->Nodes[q].data = x;
    SL->Nodes[q].link = SL.Nodes[p].link;
    SL->Nodes[p].link = q;              //插入
    return 1;
}
```

在静态链表中释放第 i 个结点

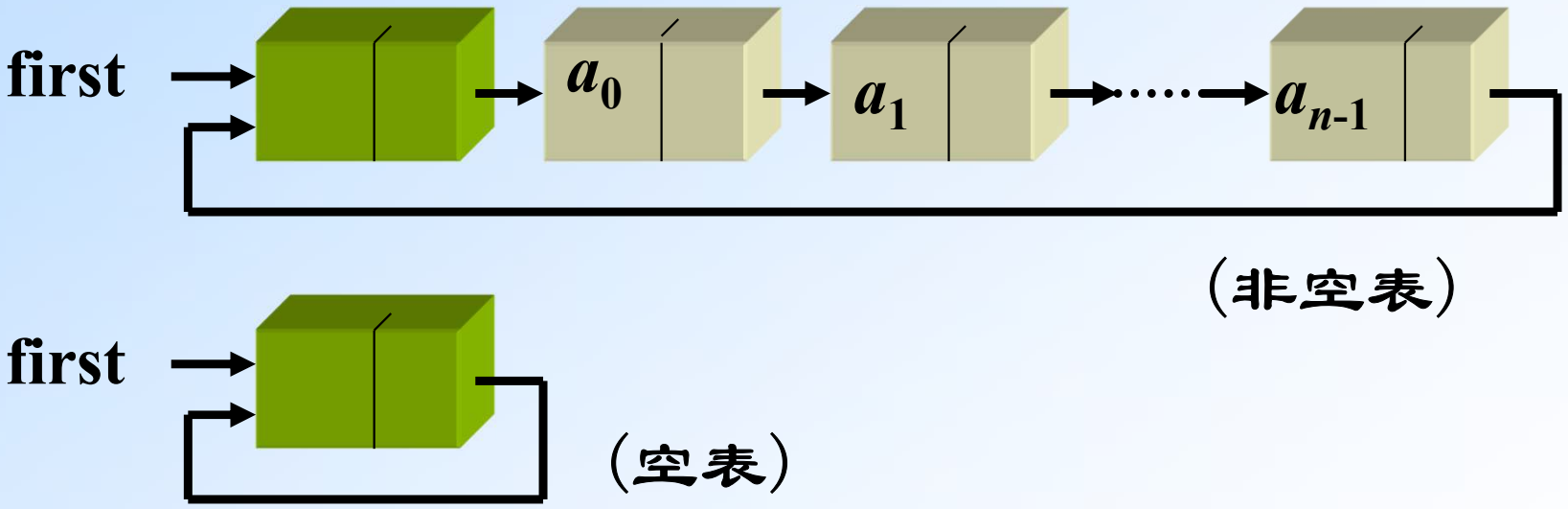
```
int Remove ( SLinkList* SL, int i ) {
    int p = Locate ( SL, i-1 );
    if ( p == -1 ) return 0;        //找不到结点
    int q = SL->Nodes[p].link;     //第 i 号结点
    SL->Nodes[p].link = SL->Nodes[q].link;
    SL->Nodes[q].link = SL->newptr;    //释放
    SL->newptr = q;
    return 1;
}
```

释放的元素放到了哪里?

循环链表 (Circular List)

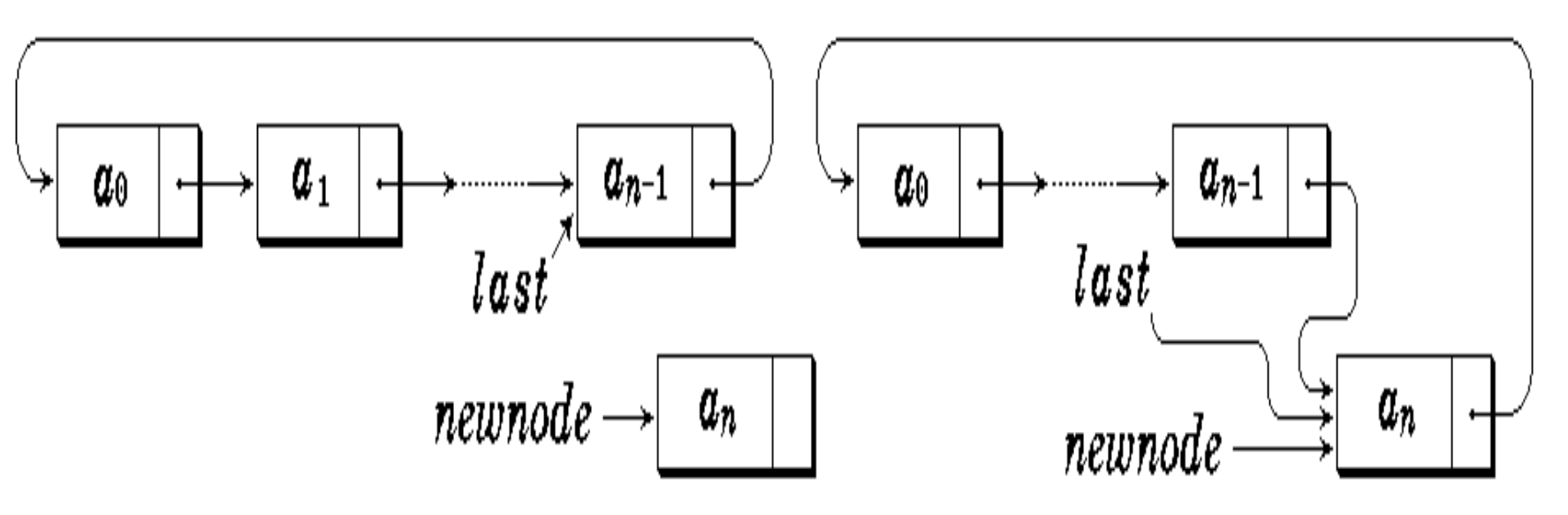
特点:最后一个结点的 link 指针不为NULL，而是指向头结点。只要已知表中某一结点的地址，就可搜寻所有结点的地址。

存储结构:链式存储结构



带表头结点的循环链表

循环链表的插入

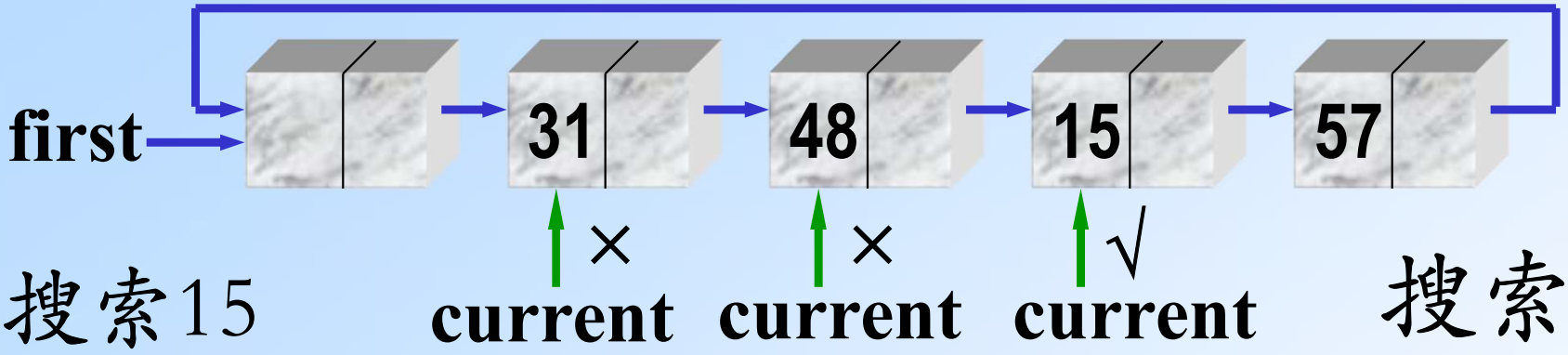


last和first的关系是什么？

循环链表类型定义

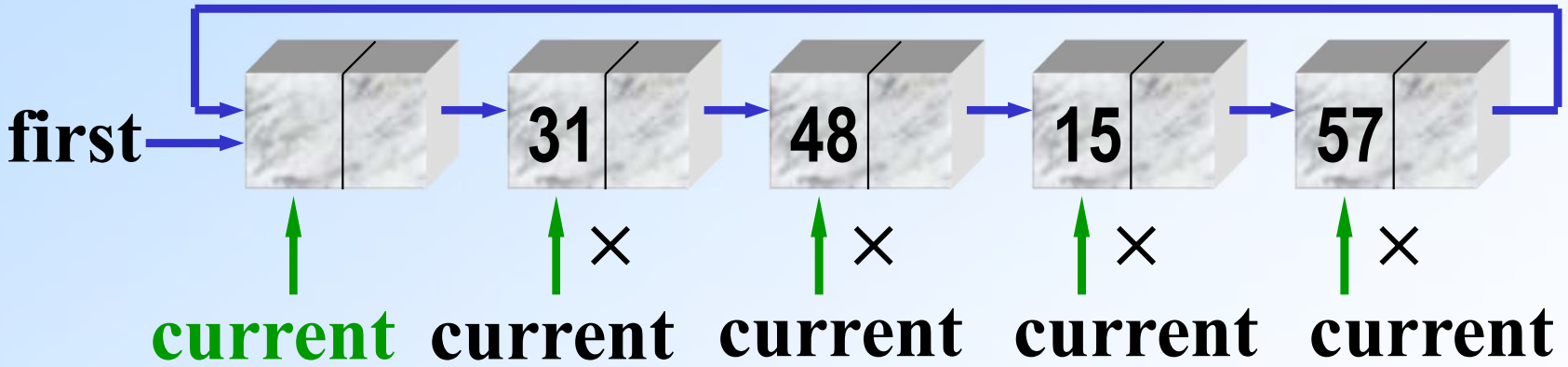
```
#define datatype int
typedef  struct node{
        datatype  data; /*数据元素类型*/
        struct node *next; /*指针类型*/
    }node, *clklist /*结点类型*/
    //其中node是节点类型
    //clklist是指向node类型的指针
```

循环链表的搜索算法



搜索15

搜索成功



搜索25

搜索不成功

循环链表的搜索算法（C描述）

```
node *find_clklist(clklist l ,int i){
        clklist p = l;
        int j = 0; /*初始化*/
        while((p->next!=l)&&(j<i)){
        /*p所指不是尾结点且还没找到时*/
                p=p->next; /*找下一个*/
                j++; /*计数器加1*/
        }
        if(j == i) return(p); /*找到第i个*/
        else return(NULL);
}
```

循环链表的搜索算法（C++描述）

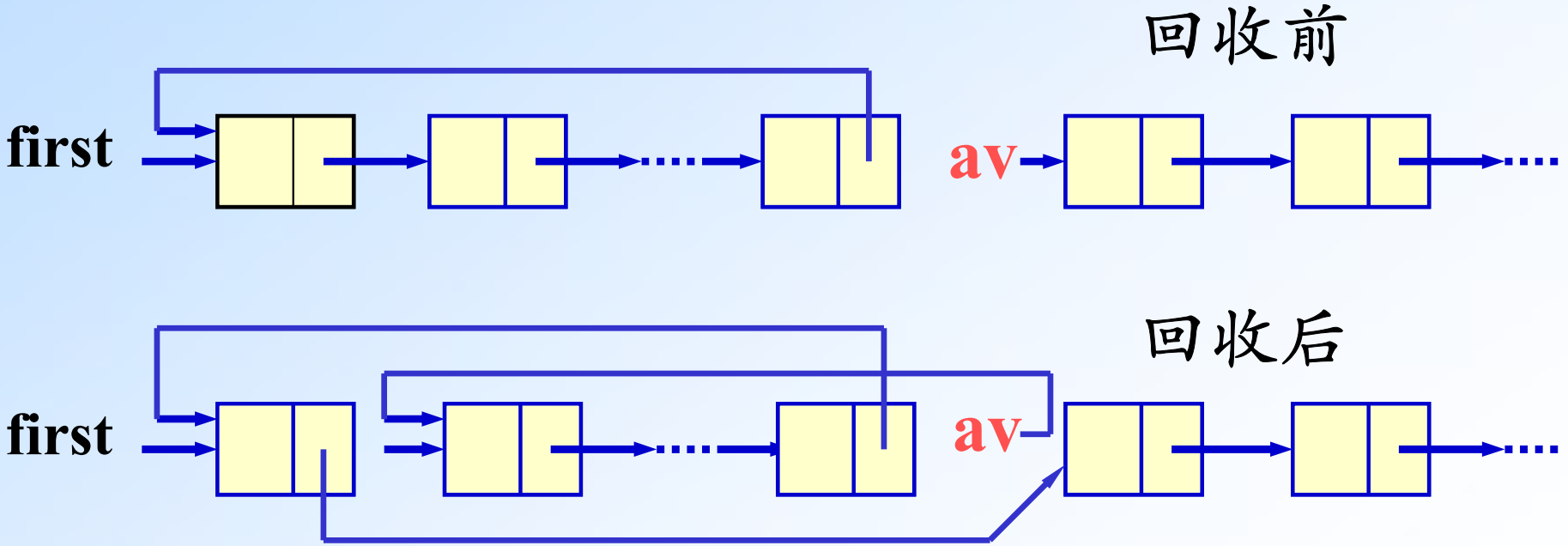
```
CircListNode * CircList::
Find ( int value ) {
//在链表中从头搜索其数据值为value的结点
    CircListNode *p = first->link;
    //检测指针 p 指示第一个结点
    while ( p != first && p->data != value )
        p = p->link;
    if(p == first) return null;
    return p;
}
```

非循环链表中在最后判断p的状态吗?

利用可利用空间表回收循环链表（C描述）

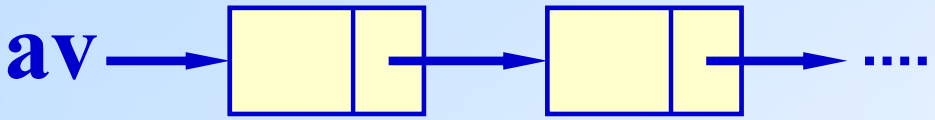
减少内存碎片

```
if ( first != NULL ) {
    CircListNode  *second = first->next;
    first-> next = av;    av = second;
    first = NULL;
}
```

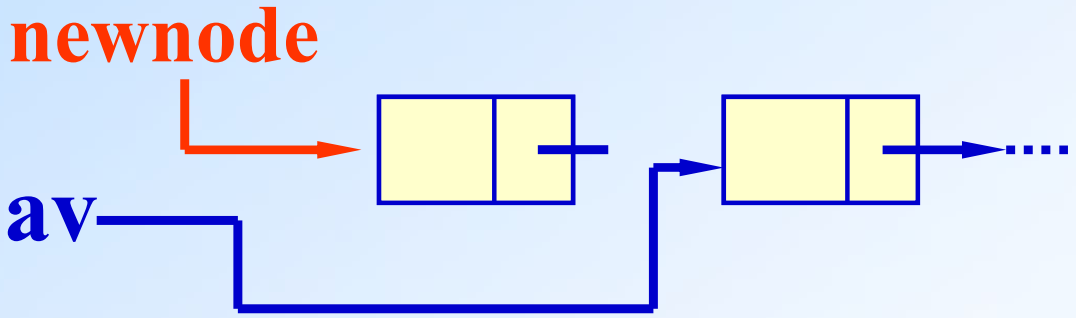


从可利用空间表分配结点（C描述）

```
if ( av ==NULL )
        CircListNode  *newnode;
else { newnode = av;  av = av->next; }
```



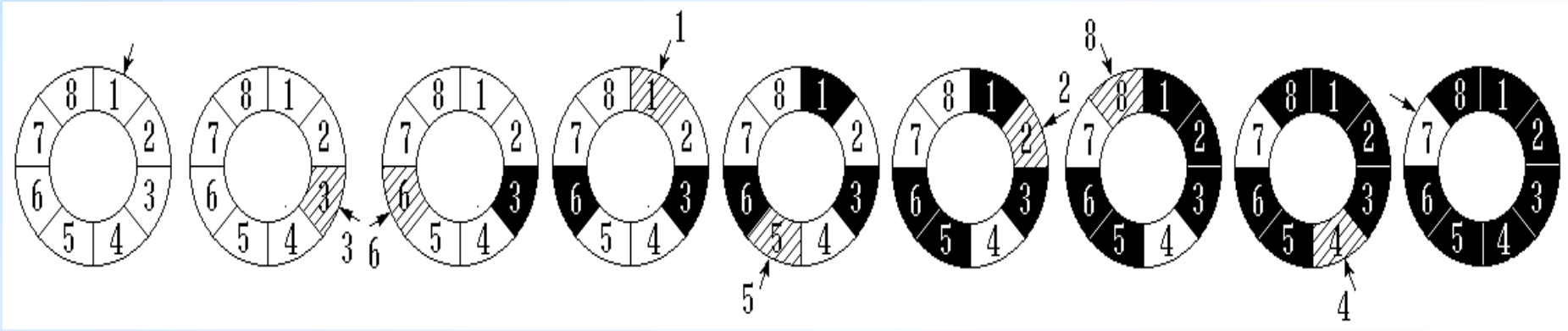
分配前的可利用空间表



分配后的可利用空间表和新结点

约瑟夫问题

- 用循环链表求解约瑟夫问题
- n 个人围成一个圆圈，首先第1个人从1开始一个人一个人顺时针报数，报到第m个人，令其出列。然后再从下一个人开始，从1顺时针报数，报到第m个人，再令其出列，…，如此下去，直到圆圈中只剩一个人为止。此人即为优胜者。
- 例如 n＝8　m＝3



约瑟夫问题的解法

```
void main ( ) {
    CircList<int> clist;
    int n, m;
    cout << “Enter the Number of Contestants?”;
    cin >> n >> m;
    for ( int i=1; i<=n; i++ )
            clist.insert (i); //形成约瑟夫环
    clist.Josephus (n, m); //解决约瑟夫问题
}
```

```
#include <iostream.h>
#include “CircList.h”
void Josephus ( int n, int m ) {
   for ( int i=0; i<n-1; i++ ) { //执行n-1次
      for ( int j=0; j<m-1; j++ )
            Next ( ); //数m-1个人
      cout << “Delete person ” << getData ( ) << endl;
      Remove ( ); //删去
   }
}
```

循环链表类的定义 (C++表示)

```
class CircList;

class CircListNode {
friend class CircList;
private:
    int data;                                   //结点数据
    CircListNode *link;       //链接指针
public:
    CircListNode ( int d = 0,
        CircListNode *next = NULL )
        : data ( d ), link ( next ) { }    //构造函数
}
```

```
class CircList {
private:
    CircListNode *first, *current, *last;
    //链表的表头指针、当前指针和表尾指针
    //哨兵结点
public:
    CircList ();
    ~CircList ( );
    int Length ( ) const;
    Boolean IsEmpty ( )
        { return first->link == first; }
    Boolean Find ( const int& value );
    int getData ( ) const;
    void Firster ( ) { current = first; }
    Boolean First ( );
    Boolean Next ( );
    Boolean Prior ( );
    void Insert ( const int& value );
    void Remove ( );
};
```

多项式 (Polynomial)

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$= \sum_{i=0}^{n} a_i x^i$$

*n*阶多项式 *Pn*(*x*) 有 *n*+1 项

- 系数 *a*0, *a*1, *a*2, ⋯, *an*
- 指数 0, 1, 2, ⋯, *n*。按升幂排列

多项式的顺序存储表示

第一种： 静态数组表示

```
private:
    int degree;
    float coef [maxDegree+1];
```

Pn(x)可以表示为：

pl.degree = n

pl.coef[i] = ai, $0 \le i \le n$

第二种：动态数组表示

```
private:
        int degree;
        float * coef;
Polynomial :: Polynomial (int sz) {
        degree = sz;
        coef = new float [degree + 1];
}
```

以上两种存储表示适用于指数连续排列的多项式。但对于指数不全的多项式如

$P_{101}(x) = 3 + 5x^{50} - 14x^{101}$, 空间浪费多

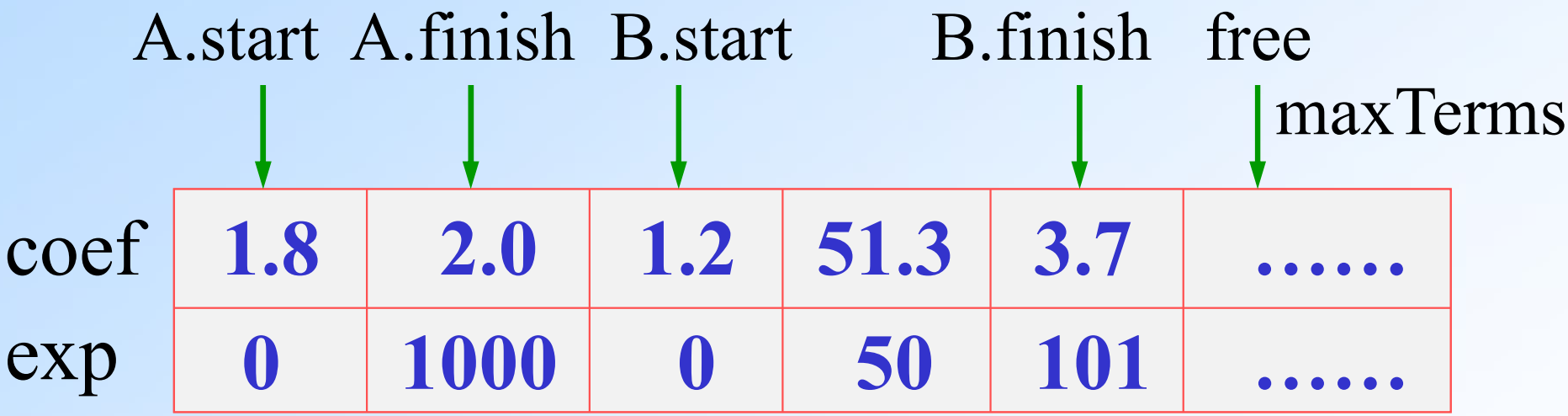
第三种：记录对应的系数和指数

```
class Polynomial;
class term {          //多项式的项定义
friend Polynomial;
private:
   float coef;        //系数
   int exp;           //指数
};
```

	0	1	2		*i*		*m*
coef	a_0	a_1	a_2	……	a_i	……	a_m
exp	e_0	e_1	e_2	……	e_i	……	e_m

$$A(x) = 2.0x^{1000}+1.8$$

$$B(x) = 1.2 + 51.3x^{50} + 3.7x^{101}$$



两个多项式存放在termArray中

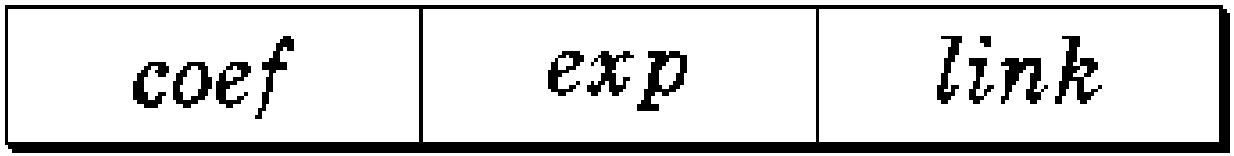
插入删除的时间复杂度？--空间换时间

还能想到别的存储方法吗？

多项式的链表表示

在多项式的链表表示中每个结点增加了一个数据成员link，作为链接指针。

$data \equiv Term$



优点是:

- 多项式的项数可以动态地增长，不存在存储溢出问题
- 插入、删除方便，不移动元素

多项式(*Polynomial*)的抽象数据类型（C）

```
typedef struct{
//项的表示，多项式的项作为LinkList的数据元素
        float coef;   //系数
        int exp;    //指数
        term *link;
}term,ElemType;
typedef  LinkList  polynomial;
//用带表头节点的有序链表表示多项式
void CreatPolyn (polynomial &P, int m);
void AddPolyn (polynomial &Pa, polynomial & Pb);
void MultiplyPolyn (polynomial &Pa, polynomial & Pb);
```

创建多项式（C语言）

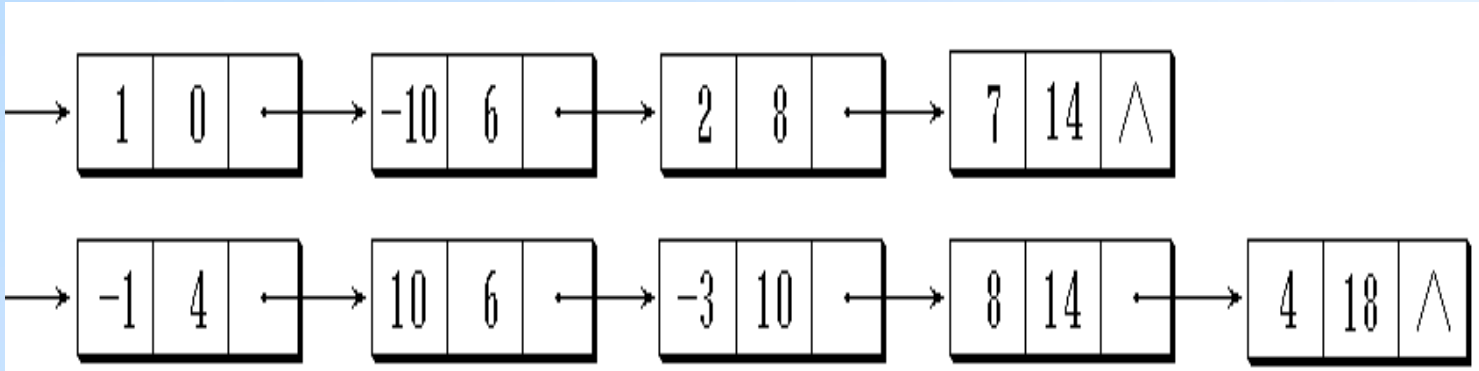
```
void CreatePolyn(polynomail &P, int m){
//输入m项的系数和指数，建立表示一元多项式的有序链表P
InitList(P);   h = GetHead(P);
//设置头结点的数据元素
term e; e.coef = 0.0;  e.expn = -1; SetCurElem(h,e);
for( i = 1;i<=m; ++1){//依次输入m个非零项
        scanf(e.coef,e.expn);
        //当前链表中不存在该指数项,函数说明参见书中
        if(!LocateElem(P,e,q,(*cmp)())){
       //依a的指数值< = >b的指数值。分别返回-1 0 1
        if(MakeNode(s,e)  InsFirst(q,s);  }//生成结点并插入链表
  }
} //CreatePolyn
```

这段代码是如何处理输入了相同指数项的？有没有更合理的方法呢？

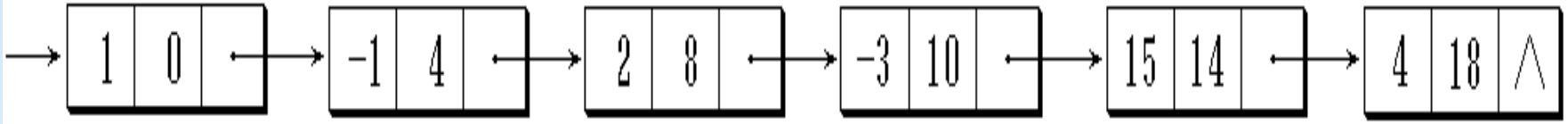
多项式链表的相加

$P1 = 1 - 10x^6 + 2x^8 + 7x^{14}$

$P2 = -x^4 + 10x^6 - 3x^{10} + 8x^{14} + 4x^{18}$



(a) 两个相加的多项式



(b) 相加结果的多项式

两个多项式的相加算法

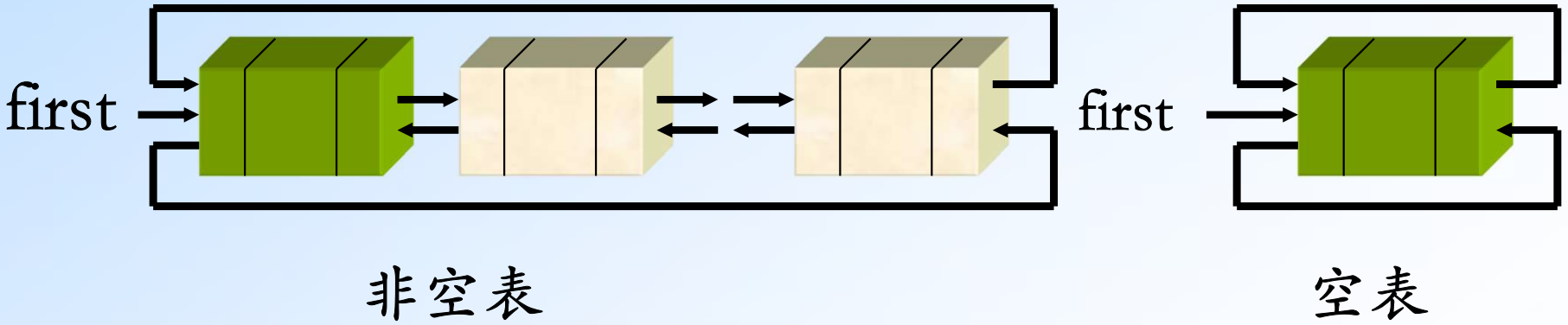
- 扫描两个多项式，若都未检测完：
  - 若当前被检测项指数相等，系数相加。若未变成0，则将结果加到结果多项式。
  - 若当前被检测项指数不等，将指数小者加到结果多项式。
- 若一个多项式已检测完，将另一个多项式剩余部分复制到结果多项式

多项式链表的相加

```
Polynomial  PolyAdd (Polynomial P1, Polynomial P2)
{
        Polynomial front, rear, temp;
        int sum;
        rear = (Polynomial) malloc(sizeof(struct PolyNode));
        front = rear;       /* 由front 记录结果多项式链表头结点 */
        while ( P1 && P2 ) /* 当两个多项式都有非零项待处理时 */
                switch ( Compare(P1->expon, P2->expon) ) {
                case 1:
                        Attach( P1->coef, P1->expon, &rear);
                        P1 = P1->link;
                        break;
                case -1:
                        Attach(P2->coef, P2->expon, &rear);
                        P2 = P2->link;
                        break;
                case 0:
                        sum = P1->coef + P2->coef;
                        if ( sum ) Attach(sum, P1->expon, &rear);
                        P1 = P1->link;
                        P2 = P2->link;
                        break;
                }
        /* 将未处理完的另一个多项式的所有节点依次复制到结果多项式中去 */
        for ( ; P1; P1 = P1->link ) Attach(P1->coef, P1->expon, &rear);
        for ( ; P2; P2 = P2->link ) Attach(P2->coef, P2->expon, &rear);
        rear->link = NULL;
        temp = front;
        front = front->link; /*令front指向结果多项式第一个非零项 */
        free(temp);          /* 释放临时空表头结点 */
        return front;
}
```

双向链表 (Doubly Linked List)

- 双向链表结点结构：



双向循环链表的定义

```
typedef int ListData;
typedef struct dnode {
    ListNode data;
    struct dnode * prior, * next;
} DblNode;
typedef DblNode * DblList;
```

建立空的双向循环链表

```
void CreateDblList ( DblList & first ) {
    first = ( DblNode * ) malloc
                ( sizeof ( DblNode ) );
    if ( first == NULL )
        { print ( “存储分配错!\n” ); exit (1); }
    //表头结点的链指针指向自己
    first->prior = first->next = first;

}
```

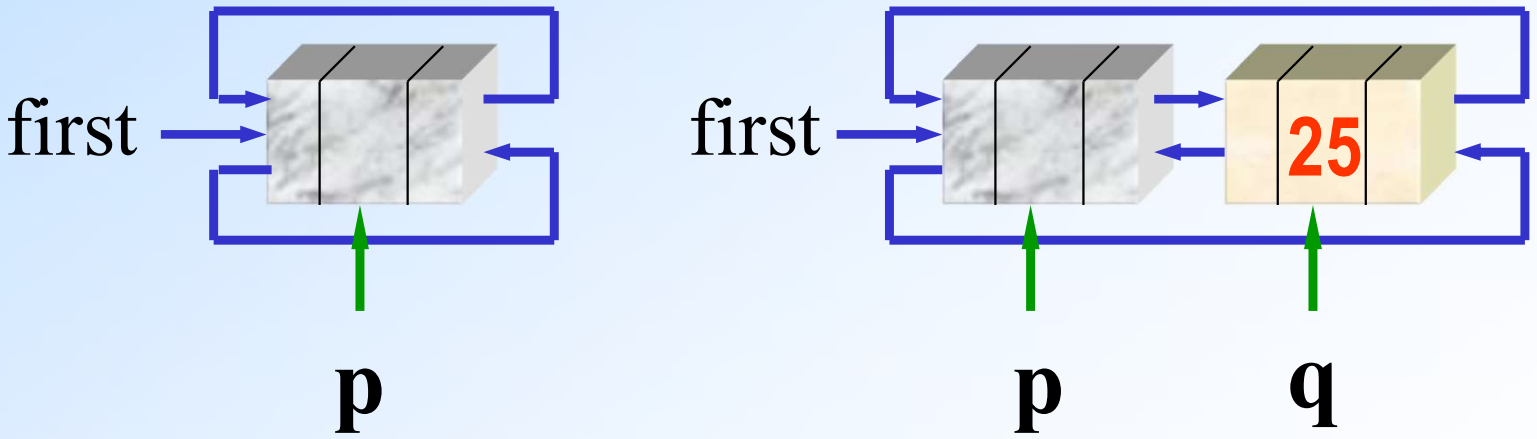
计算双向循环链表的长度

```
int Length ( DblList first ) {
//计算带表头结点的双向循环链表的长度
    DblNode * p = first->next;
    int count = 0;
    while ( p != first )
        { p = p->next;  count++; }
    return count;
}
```

双向循环链表的插入(空表)

在结点 *p 后插入25

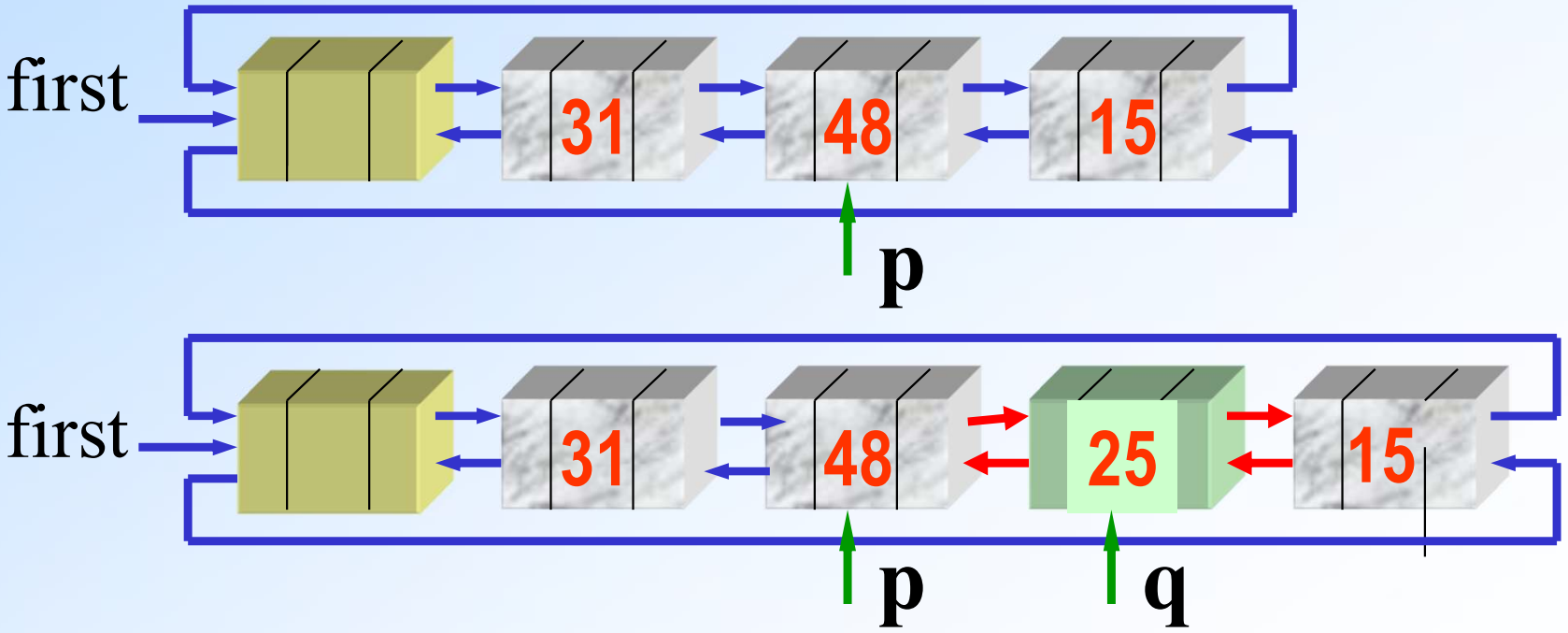
```
q->prior = p;
q->next = p->next;
 p->next = q;
q->next->prior = q;
```



双向循环链表的插入(非空表)

在结点*p后插入25

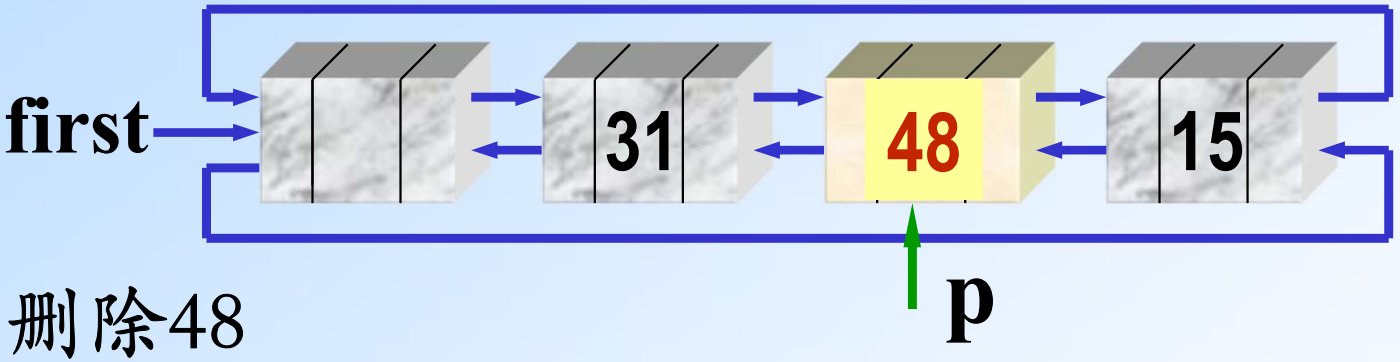
```
q->prior = p;
q->next = p->next;
p->next = q;
q->next->prior = q;
```



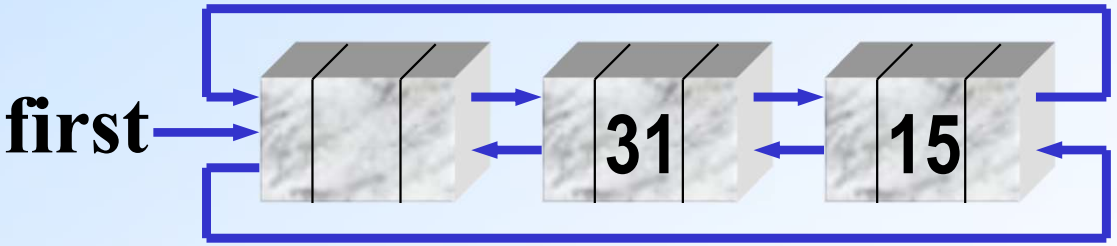
```
int Insert ( DblList first,  int i, ListData x ) {
    DblNode * p = Locate ( first, i-1 );
    //指针定位于插入位置
    if ( p == first && i != 1) return 0;
    DblNode * q = ( DblNode * ) malloc
          ( sizeof ( DblNode ) );     //分配结点
    q->data = x;
    q->prior = p;       p->next->prior = q;
    //在前驱方向链入新结点
    q->next = p->next;  p->next = q;
    //在后继方向链入新结点
    return 1;
}
```

双向循环链表的删除

p->next->prior = p->prior;
p->prior->next = p->next;

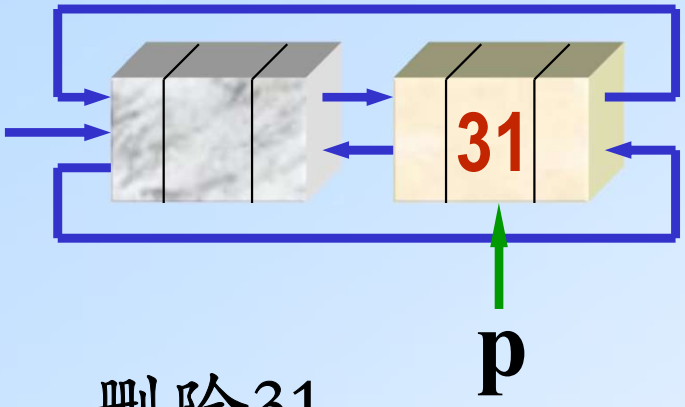


删除48

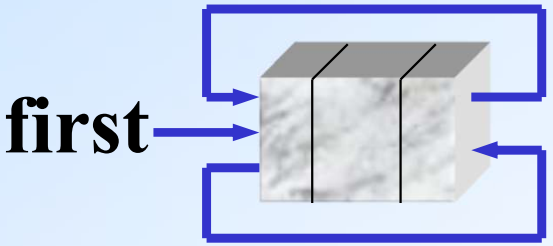


(非空表)

双向循环链表的删除



删除31




```
p->next->prior = p->prior;
p->prior->next = p->next;
```

```
int Remove ( DblList first, int i ) {
    DblNode * p = Locate ( first, i );
    //指针定位于删除结点位置
    if ( p == first ) return 0;
    p->next->prior = p->prior;
    p->prior->next = p->next;
    //删除结点 p
    free ( p );        //释放
    return 1;
}
```

课堂练习

单链表中某指针p所指结点（即p结点）的数据域为data，链指针域为next，写出在p结点之前插入s结点的操作。

课堂练习

单链表中某指针p所指结点（即p结点）的数据域为data，链指针域为next，写出在p结点之前插入s结点的操作。

设单链表的头指针为head

```
pre=head;
while(pre->next!=p)
        pre=pre->next;
s->next=p;
pre->next=s;
```

课堂练习

一元稀疏多项式以循环单链表按降幂排列，结点有三个域，系数域coef ，指数域exp和指针域next；现对链表求一阶导数，链表的头指针为ha，头结点的exp域为 －1。

```
derivative(ha){
q=ha ;  pa=ha->next;
while( (1)_____){
        if ( (2)____) {
                ( (3)__); free(pa);  pa= ( (4) _);
        }
        else{
                pa->coef ( (5) ____);
                pa->exp( (6)___);
                q=( (7) __);
        }
      pa=( (8)_____);
}}
```

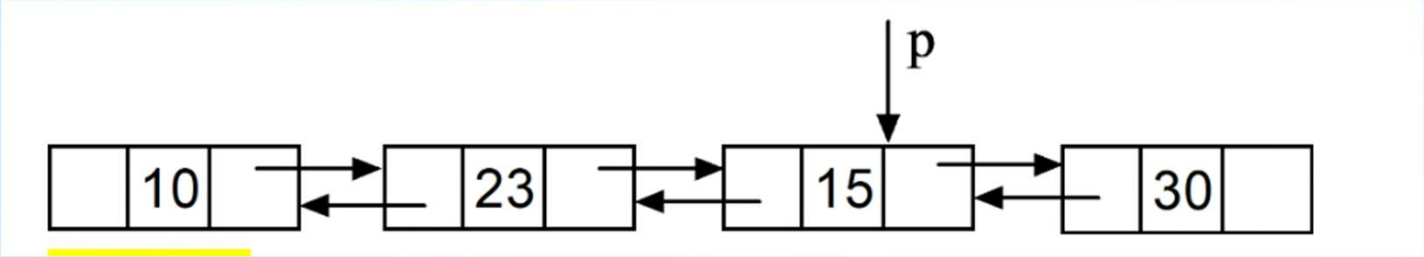
课堂练习

一元稀疏多项式以循环单链表按降幂排列，结点有三个域，系数域coef ，指数域exp和指针域 next；现对链表求一阶导数，链表的头指针为ha，头结点的exp域为 －1。

```
derivative(ha){
q=ha ;  pa=ha->next;
while( (1) pa!=ha ){
        if ( (2) pa->exp==0 ) {
                ( (3) q->next=pa->next );
                free(pa);  pa= ( (4) q );
        }
        else{
                pa->coef ( (5) =pa->coef*pa->exp );
                pa->exp( (6)-- );
                q=( (7)pa );
        }
        pa=( (8) pa->next );
}}
```

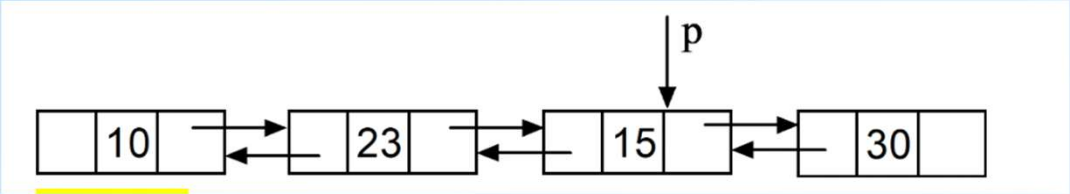
课堂练习

写出下图双链表中对换值为23和15的两个结点相互位置时修改指针的有关语句



课堂练习

写出下图双链表中对换值为23和15的两个结点相互位置时修改指针的有关语句



结点：(llink,data,rlink)

```
q=p->llink;
q->rlink=p->rlink;
p->rlink->llink=q;
p->llink=q->llink;
q->llink->rlink=p;
p->rlink=q;
q->llink=p;
```

如果我们不想这么锻炼思维，可以怎么做呢?

顺序表与链表的比较

基于空间的比较

- 存储分配的方式
  - 顺序表的存储空间是静态分配的
  - 链表的存储空间是动态分配的
- 存储密度＝结点数据本身所占的存储量/结点结构所占的存储总量
  - 顺序表的存储密度＝1
  - 链表的存储密度＜1

顺序表与链表的比较

基于时间的比较

- 存取方式
  - 顺序表可以随机存取，也可以顺序存取
  - 链表是顺序存取的
- 插入/删除时移动元素个数
  - 顺序表平均需要移动近一半元素
  - 链表不需要移动元素，只需要修改指针