动态变量的应用—动态数组

- 对输入的若干个数进行排序，如果输入时先输入数的个数，然后再输入各个数，可用动态数组表示这些数：

```
int n;
int *p;
cin >> n;
p = new int[n];
for (int i=0; i<n; i++)
      cin >> p[i];
sort(p,n);
……
delete []p;
```

对输入的若干个数进行排序，在输入时，先输入各个数，最后输入一个结束标记（如：-1），这时，可以按以下方式实现：

```
const int INCREMENT=10;
int max_len=20,count=0,n,*p=new int[max_len];
cin >> n;
while (n != -1)
{  if (count >= max_len)
   {     max_len += INCREMENT;
         int *q=new int[max_len];
         for (int i=0; i<count; i++) q[i] = p[i];
         delete []p;
         p = q;
   }
   p[count] = n;
   count++;
   cin >> n;
}
sort(p,count);
……
delete []p;
p= NULL;
```

开始不知道数据规模，因此预设一定量，
但当预设的数值仍不能满足要求时☹，
需要对数组“扩长”，牵涉“数据转移”☹…
如何解决该问题？

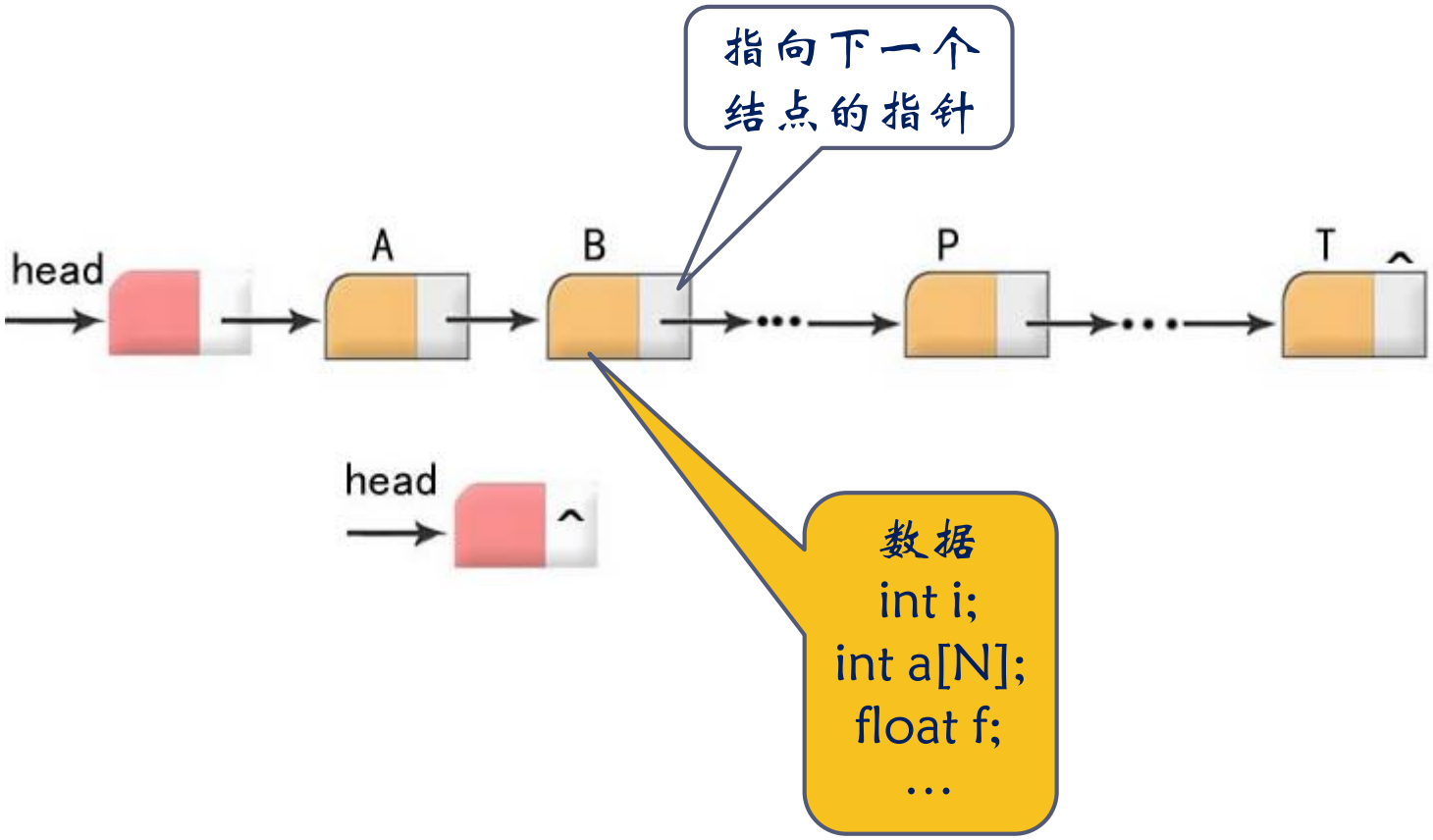
6 复杂数据的描述—构造数据类型
6.6 链　表

郭延文

2019级计算机科学与技术系

- 以上例题对于“不确定个数”输入数据排序的实现方法虽然可行，但是：
  - 当数组空间不够时，需要重新申请空间、进行数据转移以及释放原有的空间（切记！），这样做比较麻烦并且效率有时不高
  - 当需要在数组中增加或删除元素时，还将会面临数组元素的大量移动问题。
- 链表可以避免数组的上述问题

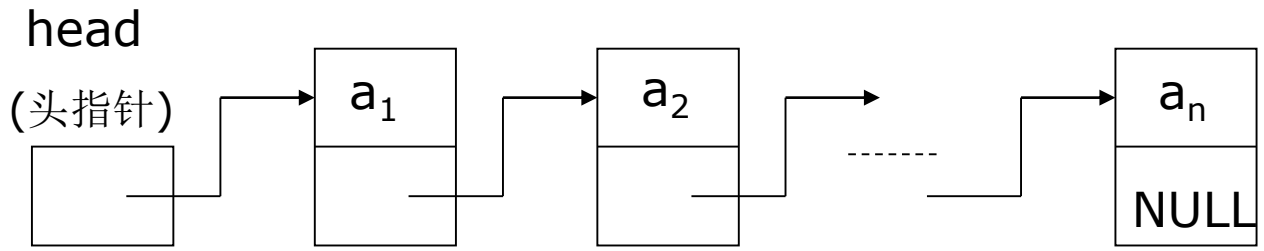
链 表



动态变量的应用——链表

- 链表用于表示由若干个同类型的元素所构成的具有线性结构的复合数据。
- 链表元素在内存中不必存放在连续的空间内。
- 链表中的每一个元素除了本身的数据外，还包含一个（或多个）指针，它（们）指向链表中下一个（前一个或其它）元素。
- 如果每个元素只包含一个指针，则称为单链表，否则称为多链表。

单链表



- 单链表的每个元素只包含一个“指向相邻节点”的指针。
- 需要一个头指针，指向第一个元素。
- 单链表中的结点类型和表头指针变量可定义如下：

```
struct Node                 //结点的类型定义
{     int content;          //代表结点的数据
      Node *next;           //代表后一个结点的地址
};
Node *head=NULL; //头指针变量定义，初始状态下
                 //为空值。NULL在cstdio中定义为0
```

链表操作

基本操作：

- 链表的建立
- 链表的遍历和元素定位（头、尾、特定位置和特定值）
- 链表中节点的插入（头、尾、特定位置）与删除
- 链表的输出与删除

衍生操作：

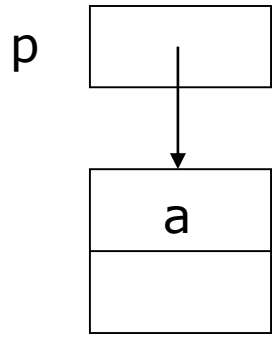
- 链表的反转
- 基于链表的排序
- 特定元素检索程序
- …

基本操作：
在链表中插入一个结点

- 首先产生一个新结点：

```
Node *p=new Node;    //产生一个动态变量来表示新结点
p->content = a;      //把a赋给新结点中表示结点值的成员
```

- 图示为：



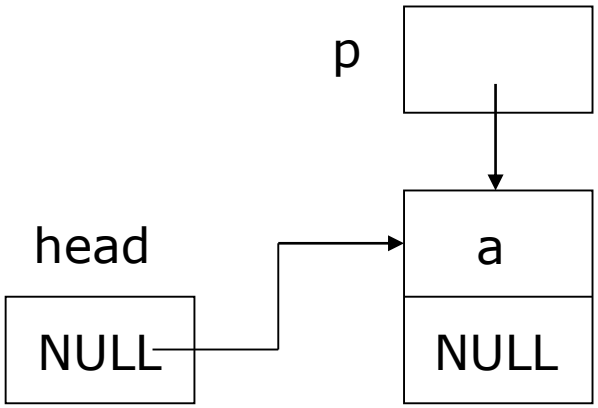
该结点插入在什么位置？

- 表头（链表头部）？
- 表尾（链表尾部）？
- 第i个结点的后面
- 该结点是否是链表的第一个结点（此前为空）？

- 如果链表为空（创建第一个结点时），则进行下面的操作：

```
if (head == NULL)        //表头指针为空
{
        head = p;                  //头指针指向新结点。
        p->next = NULL;            //等价于head->next = NULL;
                                   //把新结点的next成员置为NULL。
}
```

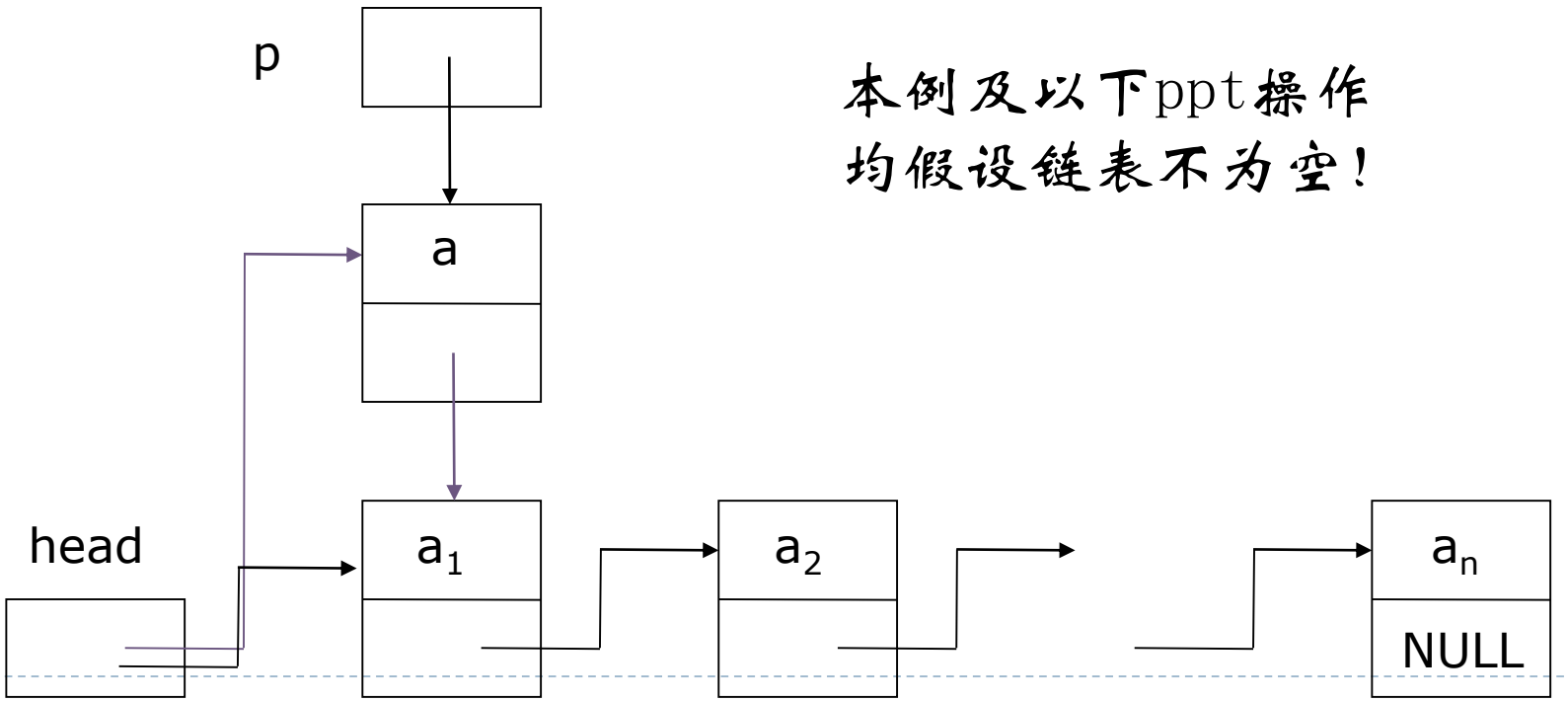
- 图示为：



- 如果新结点插在表头，则进行下面的操作：

```
p->next = head;   //把新结点的下一个结点指定为
                  //链表原来的第一个结点。
head = p;         //表头指针指向新结点
```

- 图示为：

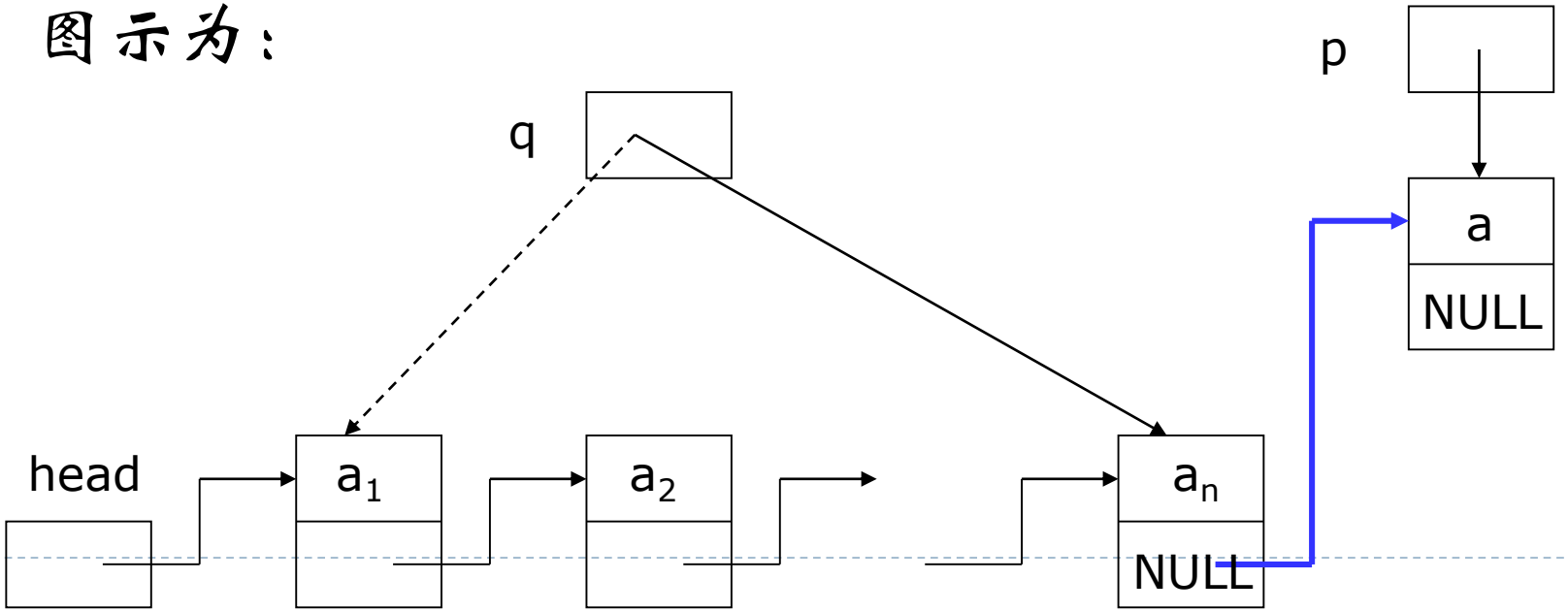


本例及以下ppt操作
均假设链表不为空！

- 如果新结点插在表尾，则进行下面的操作：

```
Node *q=head;                    //q指向第一个结点
while (q->next != NULL)  //循环查找最后一个结点
    q = q->next;     //循环结束后，q指向链表最后一个结点
q->next = p;                     //把新结点加到链表的尾部
p->next = NULL;                  //把新结点的next成员置为NULL
```

- 图示为：



- 如果新结点插在链表中第i（i>0）个结点（a_i）的后面，则进行下面的操作：

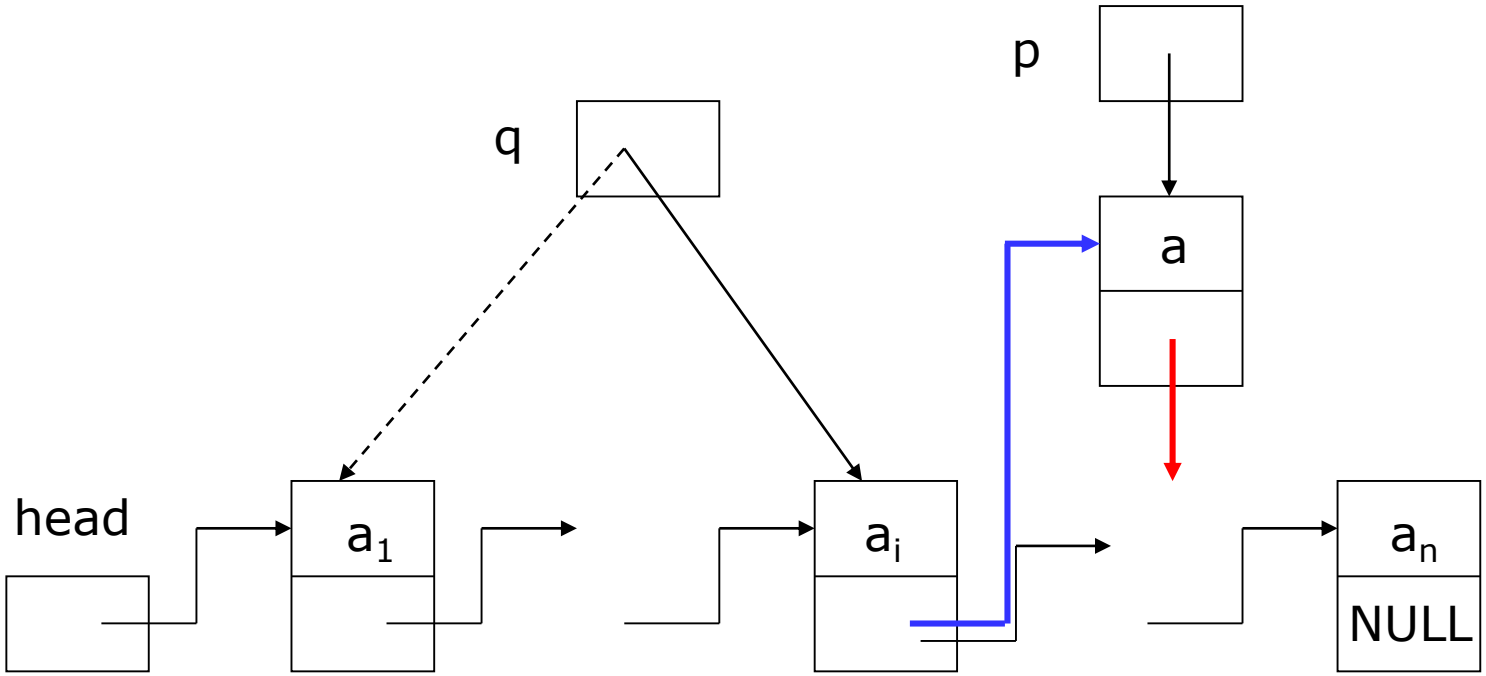
```
Node *q=head; //q指向第一个结点。
int j=1; //当前结点的序号，初始化为1
while (j < i && q->next != NULL) //循环查找第i个结点
{    q = q->next;                    //q指向下一个结点
     j++;                            //结点序号增加1
}
//循环结束时，q或者指向第i个结点，或者指向最后一个结点（结点数
   不够i时）。
if (j == i) //q指向第i个结点。
```

?

```
else //链表中没有第i个结点。
     cout << "没有第" << i << "个结点\n";
```

```
p->next = q->next;      // 把q所指向结点的下一个结点指定为
                        // 新结点的下一个结点
q->next = p;            // 把新结点指定为q所指向结点的下一个结点
```

- 图示为：



- 如果新结点插在链表中第i（i>0）个结点（a_i）的后面，则进行下面的操作：

```
Node *q=head; //q指向第一个结点。
int j=1; //当前结点的序号，初始化为1
while (j < i && q->next != NULL) //循环查找第i个结点
{    q = q->next;                    //q指向下一个结点
     j++;                            //结点序号增加1
}
//循环结束时，q或者指向第i个结点，或者指向最后一个结点（结点数
   不够i时）。
if (j == i) //q指向第i个结点。
{    p->next = q->next;   //把q所指向结点的下一个结点指定为
                          //新结点的下一个结点
     q->next = p;         //把新结点指定为q所指向结点的下一个结点
}
else //链表中没有第i个结点。
     cout << "没有第" << i << "个结点\n";
```

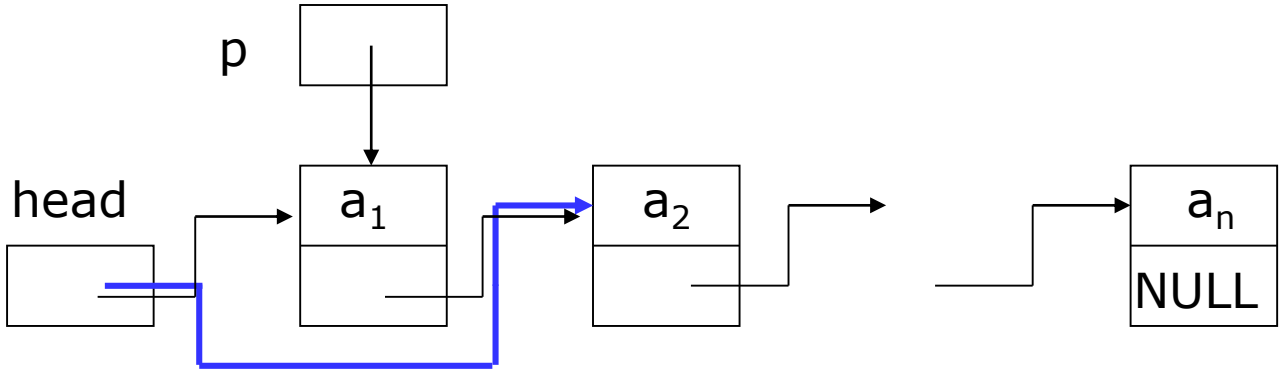
基本操作：
在链表中删除一个结点

下面的操作假设链表不为空，即：head != NULL

- 如果删除链表中第一个结点，则进行下面的操作：

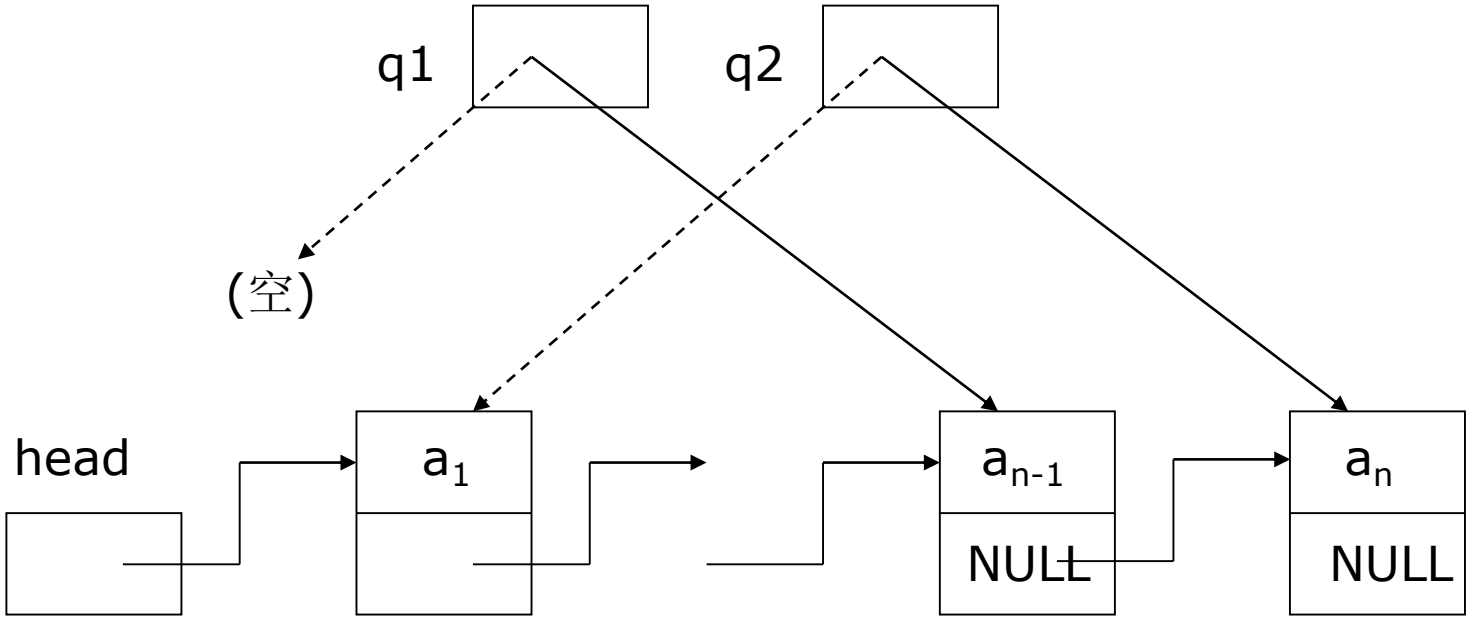
```
Node *p=head; //p指向第一个结点。
head = head->next;  //头指针指向第一个结点的下一个结点。
delete p;  //归还删除结点的空间。
```

- 图示为：



如果删除链表的最后一个结点

- 图示为：



- 如果删除链表的最后一个结点，则进行下面的操作：

```
Node *q1=NULL,*q2=head;
//循环查找最后一个结点，找到后，q2指向它，q1指向它的前一个结点。
while (q2->next != NULL)
{  q1 = q2;
   q2 = q2->next;
}
if (q1 == NULL)      //链表中只有一个结点。
   head = NULL;      //把头指针置为NULL。
else //存在倒数第二个结点。
   q1->next = NULL; //把倒数第二个结点的next置为NULL。
delete q2;           //归还删除结点的空间。
```

- 如果删除链表中第i（i>0）个结点a_i，则进行下面的操作：

```
if (i == 1) //要删除的结点是链表的第一个结点。
{  Node *p=head;          //p指向第一个结点。
   head = head->next;  //head指向第一个结点
                          //的下一个结点。
   delete p;              //归还删除结点的空间。
}
```

```
else                                    //要删除的结点不是链表的第一个结点。
{  Node *p=head;          //p指向第一个结点。
   int j=1; //当前结点的序号，初始化为1
   while (j < i-1)                      //循环查找第i-1个结点。
   {    if (p->next == NULL)
            break;                      //当没有下一个结点时，退出循环
        p = p->next;      //p指向下一个结点
        j++; //结点序号加1
   }
   if (p->next != NULL)                 //链表中存在第i个结点。
   {    Node *q=p->next;     //q指向第i个结点。
        p->next = q->next;   //把第i-1个结点的next
                             //改成第i个结点的next
        delete q;                       //归还第i个结点的空间。
   }
   else //链表中没有第i个结点。
        cout << "没有第" << i << "个结点\n";
}
```

在链表中检索某个值a

```
int index=0;    //用于记住结点的序号，初始化为0
//从第一个结点开始遍历链表的每个结点查找值为a的结点。
for (Node *p = head; p! = NULL; p = p->next)
{  index++;     //记住结点的序号，下面输出时需要。
   if (p->content == a)
            break;
}
if (p != NULL) //找到了
   cout << "第" << index << "个结点的值为：" << a << endl;
else //未找到
   cout << "没有找到值为" << a << "的结点\n";
```