> 链表恶心在于所有涉及头、尾的操作都要单独考虑

1、声明类型

```
struct node
 {
        int content;
    char str【10】;
        node *next;
 };
```

2、输入数据

表尾插入法

```
node *input1()//*别忘
{
node *head=NULL,*tail;//*别忘
int x;
cin>>x;
while(x!=-1)
     {
      node *p=new node;
      p->content=x;
     p->next=NULL;
     if(head==NULL) head=tail=p;
        //如果是空表,加入第一个结点
        else
          {
          tail->next=p;
          //新结点加在表尾
          tail=p;//表尾指针指向新结点
          }
     cin>>x;
    }
return head;
}
```

- 返回首结点的函数用node *
- 插入新结点，一定先定义动态变量指针

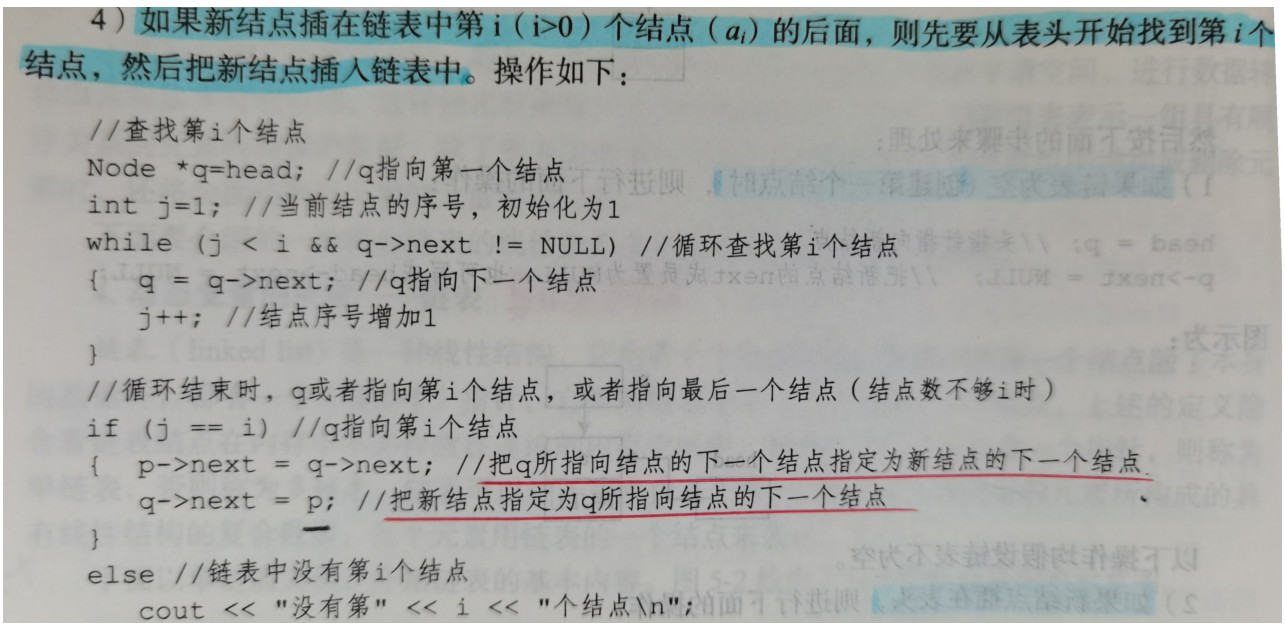
- head特殊，总需要单独讨论
- 不要忘最后一位必须NULL
- 理解结点赋值：=后面的位置给=前面的

表头插入法

```
node *input2()
{
node *head=NULL;
int x;
cin>>x;
while(x!=-1)
    {
    node *p=new node;
    p->content=x;
   p->next=head;
   //新结点插入表头 ，原来的head结点变成第二个
   head=p;//把新结点设置为表头结点
   cin>>x;
   }
return head;
}
```

插在i位后

> 先查找第i位



4）如果新结点插在链表中第 i（i>0）个结点（a_i）的后面，则先要从表头开始找到第 i 个结点，然后把新结点插入链表中。操作如下：

```
//查找第i个结点
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";
```

插入i位后函数

> h表头指针，a要插入的结点值
> 当i为0表示在表头插入
> 操作成功返回true，否则返回false

```
bool in(Node *&h,int a,int i)
{
 Node *q=new Node(a);
 //创建结点时直接赋值a
 if (i==0)
   {
    q->next=h;
    h=q;
    return true;
    }
 else
   {
    Node *p=h;
    int j=1;
    while (p!=NULL && j<i)
     {
       p=p->next;
       j++;
      }
    if (p!=NULL)
      {
        q->next=p->next;
        p->next=q;
        return true;
       }
    else
      return false;
   }
}
```

3、结点定位

r的前一个

```
node *u=new node;
u=head1;
while(u->next!=r)
    u=u->next;
```

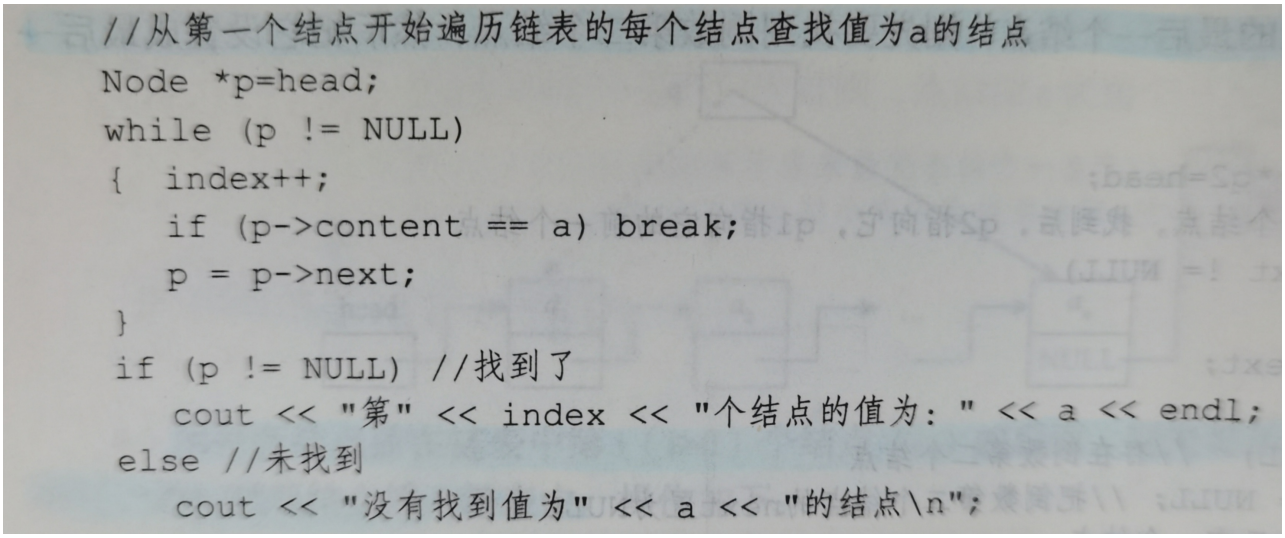
查找第i位

```
//查找第i个结点
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）
```

最后一个结点

```
node *u=new node;
u=head1;
while(u->next!=NULL)
  u=u->next;
```

查找对应值



```
//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
Node *p=head;
while (p != NULL)
{  index++;
   if (p->content == a) break;
   p = p->next;
}
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";
```

4、输出结点

```
void output(node *h)
{
for(node *p=h;p!=NULL;p=p->next)
   cout<<p->content<<" ";
cout<<endl;
}
```

5、删除结点

删除第i位函数

```
bool remove(Node *&h,int &a, int i)
{
 Node *p=h, *q=null;
 int j=1;
 while (p!=NULL && j<i)
   {
    q=p;
    p=p->next;
    j++;
```

```
    }
if (p!=NULL)
    {
    a=p->jd;
    if (q!=NULL)
        q->next=p->next;
    else
        h = p->next;
    delete p;
    return true;
 }
 else
     return false;
}
```

删除函数

```
void remove(node *h)
{
for(node *p=h;h!=NULL;h=h->next)
    delete p;
}

```

- head和tail往往需要单独考虑

6、main函数

```
head1=input1();//插入第一个链表
head3=func（head1,head2）;
//函数操作完要赋值给head3才有效！！
```

7、链表反转

循环反转

> 每三个一组，只要箭头反过来就可以

```
Node * Reverse(Node *head)
{
Node *prev = NULL;
Node *cur = NULL;
Node *next = head;
while(next != NULL)
  {
  prev = cur;
  cur = next;
 next = cur -> next;
 //prev、cur、next后移一位
 cur -> next = prev;
 //反转方向使cur指向prev
 //（原next指向cur）
 }
return cur;
}
```

递归反转

> 理解方式：结点不动，箭头动

```
Node *RecReverse(Node *head)
{
if(!head) //整个链表为空
  return NULL;

Node *temp = RecReverse
(head -> next);//反转小规模链表
if(!temp) //小规模链表为空
  return head;

head -> next -> next = head;
```

```
 //head->next本来是小规模链表的尾
 //让head->next指向head
 //head变成小规模链表尾
head -> next = NULL;
//小规模链表的尾部置空

return temp;//返回小规模链表首节点地址
}
```

8、环链表

- 完成闭环，自动删除中间结点

```
//r在q前，p在q后
r->next=p;

```

> 最恶心：处理第一步，处理收尾
> 小心多个p，q，i，j定义混乱

- 查找前第k个结点

> 需要向后查n-i-k个
> fabs（）才对！n-i-k可以为负