西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C++程序设计
Programming in C++

1011018

主讲：魏英，计算机学院

自定义数据类型的应用——链表

3、链表的运算

- （1）链表遍历ListTraverse(L,visit())
- 与数组不同，链表不是用下标而是用指针运算查找数据元素的。通过链表的头指针L可以访问开始结点p=L->next，令p=p->next，即p指向直接后继结点，如此循环可以访问整个链表中的全部结点。
- 链表遍历算法的实现步骤为：
- ①令指针p指向L的开始结点。
- ②若p为0，表示已到链尾，遍历结束。
- ③令p指向直接后继结点，即p=p->next。重复②～③步骤直至遍历结束。

- 链表遍历的算法如下：

```
1  void ListTraverse(LinkList L,
         void(*visit)(ElemType*))
2  {   //遍历L中的每个元素且调用函数visit访问它
3      LinkList p=L->next; //p指向开始结点
4      while(p!=NULL) { //若不是链尾继续
5          visit(&(p->data));
6          p=p->next; //p指向直接后继结点
7      }
8  }
```

- 其中visit是函数指针。当调用ListTraverse遍历结点时，通过调用visit()对每个结点完成定制的操作。

```
1  void visit(ElemType *ep) //实现链表遍历时结点访问的定制函数
2  {    //在函数中对结点*ep实现定制的操作，例如输出
3       cout<<*ep<<" ";
4  }
```

- （2）查找结点，返回链表中满足指定数据元素的位序 LocateElem(L,e,compare())
- 应用遍历算法查找链表结点，返回第一个满足定制关系数据元素的位序的算法如下：

```
1   int LocateElem(LinkList L,ElemType e,
        int(*compare)(ElemType*,ElemType*))
2   {//返回L中第1个与e满足关系compare()的元素的位序
3       int i=0;
4       LinkList p=L->next; //p指向开始结点
5       while(p!=NULL) { //若不是链尾继续
6           i++; //记录结点的位序
7           if(compare(&(p->data),&e))  return i;
8           p=p->next; //指向直接后继结点
9       }
10      return 0; //关系不存在返回0
11  }
```

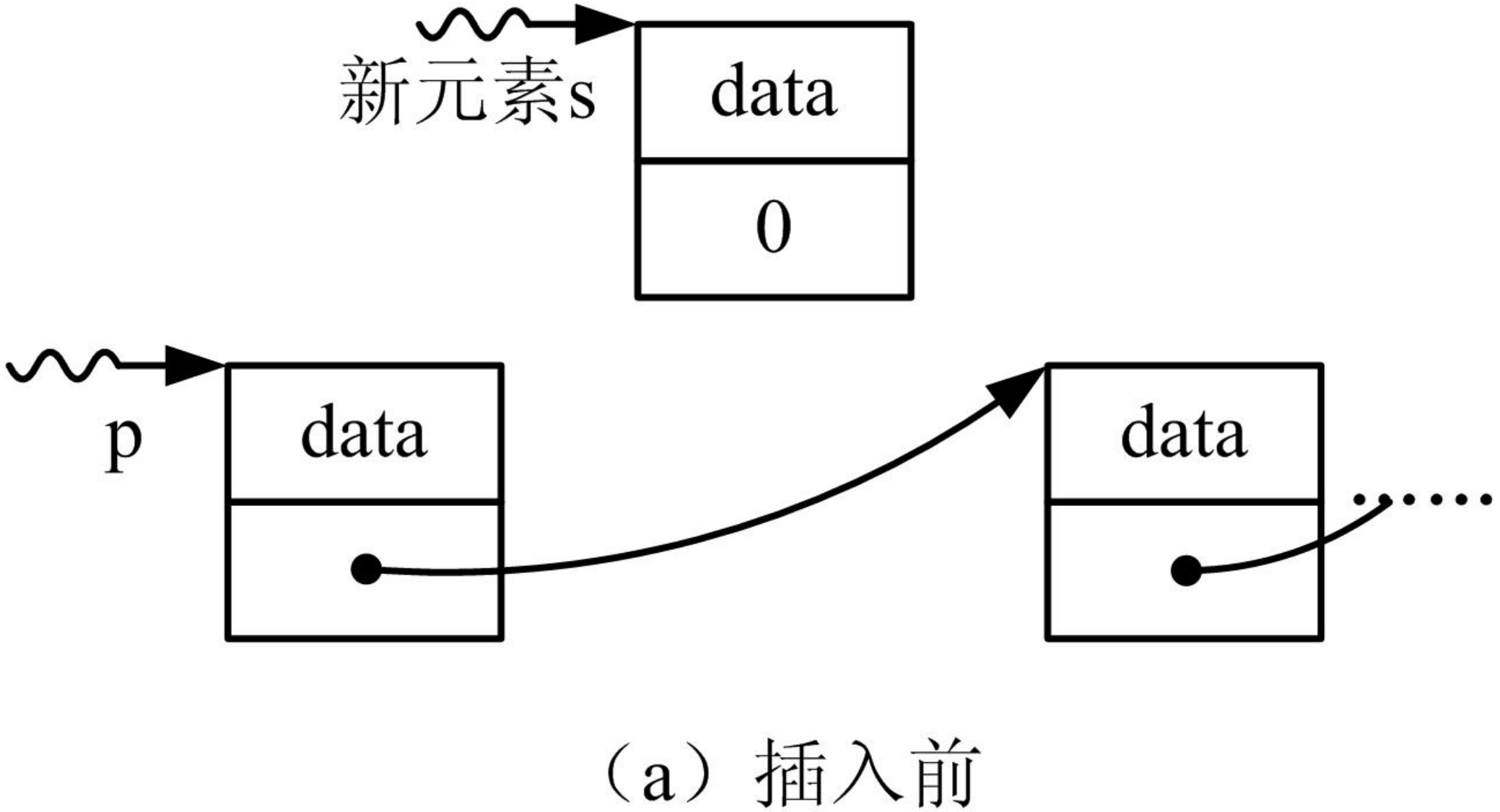
- 其中compare是函数指针。当调用LocateElem遍历结点时，通过调用compare()对每个结点与给定完成定制的关系比较，关系成立返回真，否则返回假。如相等比较为

```
1  int compare(ElemType *ep1,ElemType *ep2)
   //实现两个数据元素关系比较的定制函数
2  {//在函数中对数据元素进行定制的关系比较，如相等，大于或小于
3      if (*ep1==*ep2) return 1; //满足相等关系返回真（1）
4      return 0; //不满足关系返回假（0）
5  }
```

- （3）插入结点
- 插入结点操作是指将一个新结点插入到已知的链表中。插入位置可能在头结点、尾结点或者链表中间，插入操作前需要定位插入元素的位置和动态分配产生新结点。

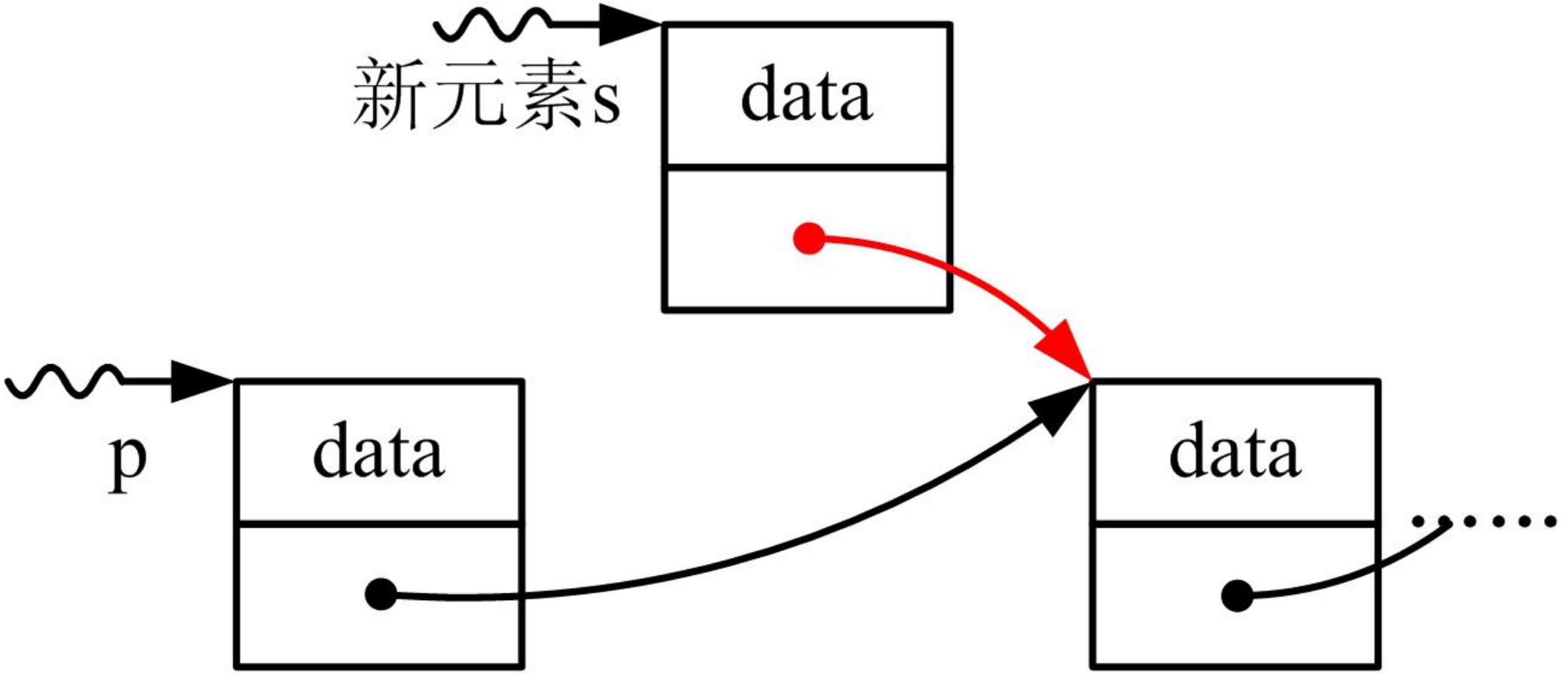
- 假设将新结点s插入到单链表的第i个结点位置上。方法是先在单链表中找到第i-1个结点p，在其后插入新结点s。

图22.4 单链表插入结点示意



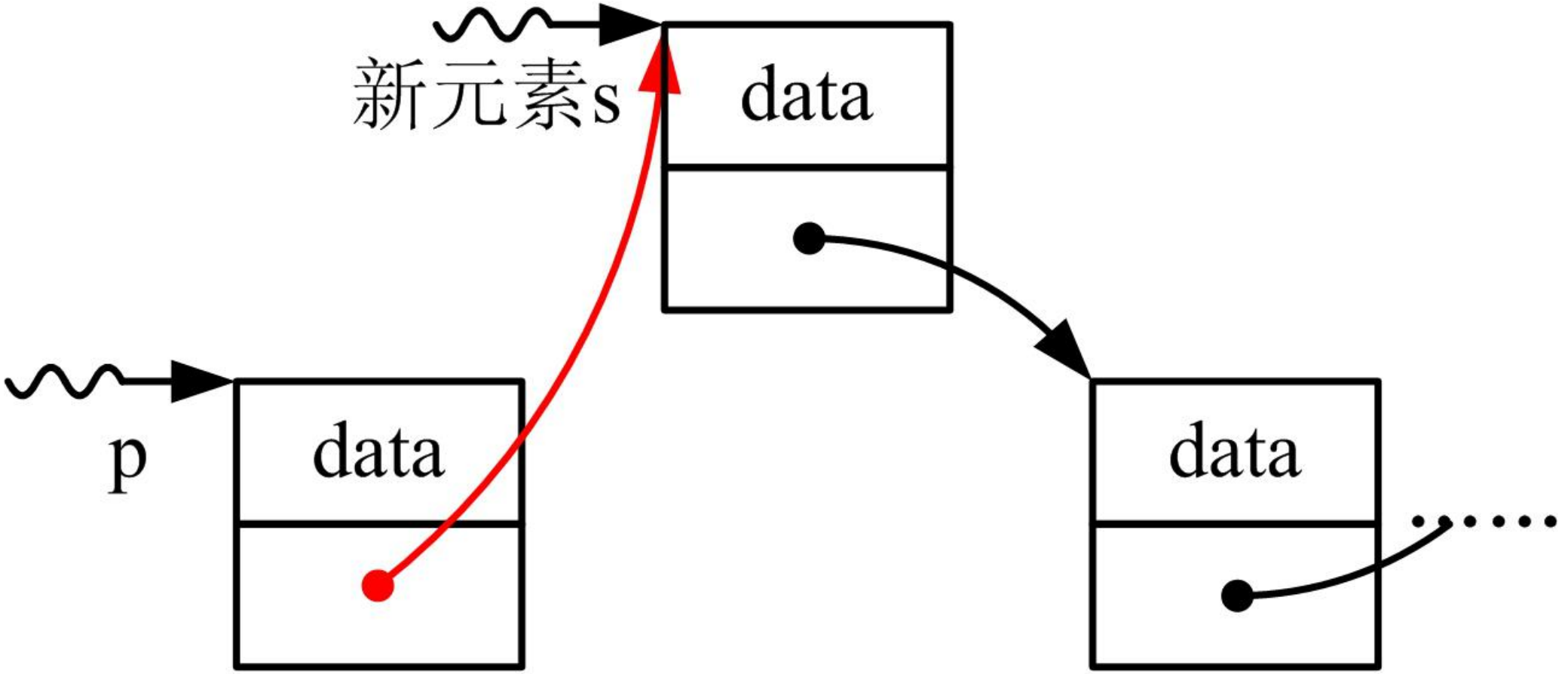
（a）插入前

图22.4 单链表插入结点示意



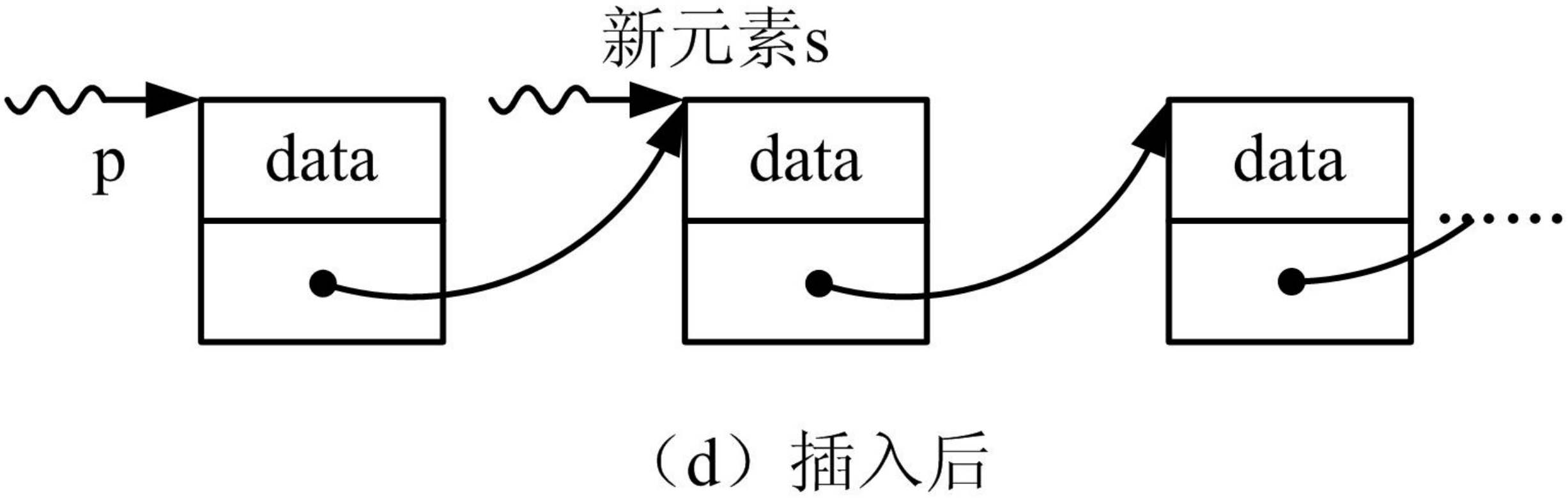
（b） s->next=p->next

图22.4 单链表插入结点示意



（c） p->next=s

图22.4 单链表插入结点示意



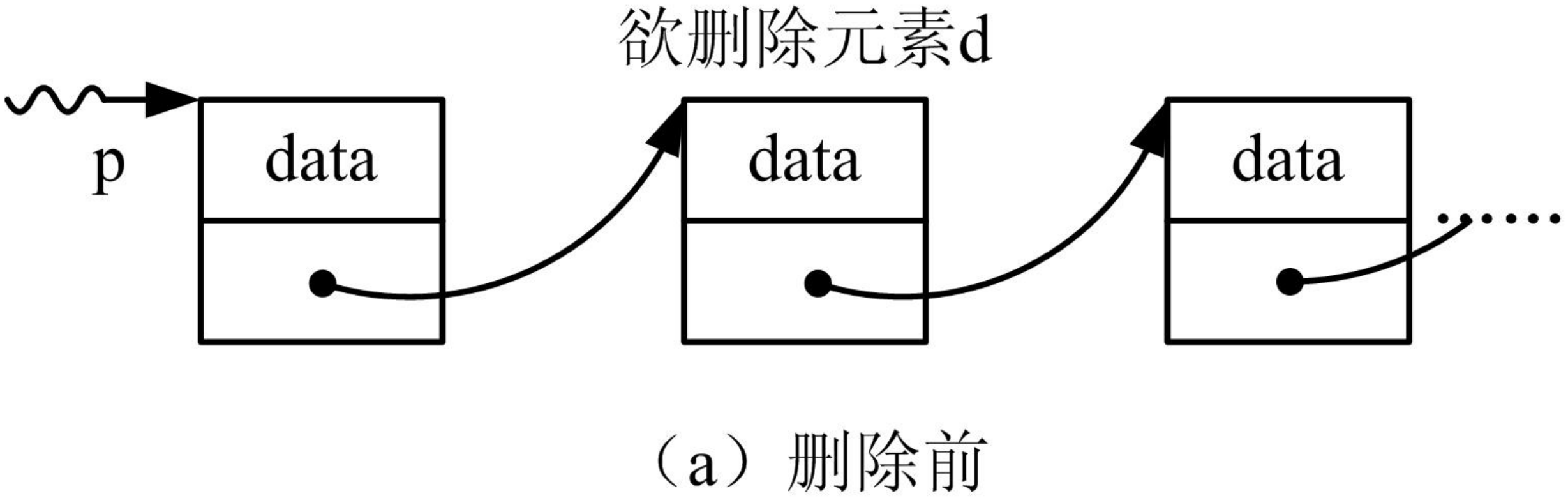
（d）插入后

```
1  int ListInsert(LinkList *L,int i,ElemType e)
2  { //在第i个位置之前插入元素e
3    LinkList s,p=*L; //p指向头结点
4    while(p!=NULL && i>1) { //寻找第i-1个结点
5      p=p->next; //p指向直接后继结点
6      i--;
7    }
8    if(p==NULL||i<1) return 0; //i值不合法返回假（0）
9    s=new LNode; //创建新结点
10   s->data=e; //插入L中
11   s->next=p->next, p->next=s; //结点插入算法
12   return 1; //操作成功返回真（1）
13 }
```

- （4）删除结点
- 结点删除操作是指将链表中的某个结点从链表中删除。删除位置可能在头结点、尾结点或者链表中间，删除操作后需要释放删除结点的内存空间。

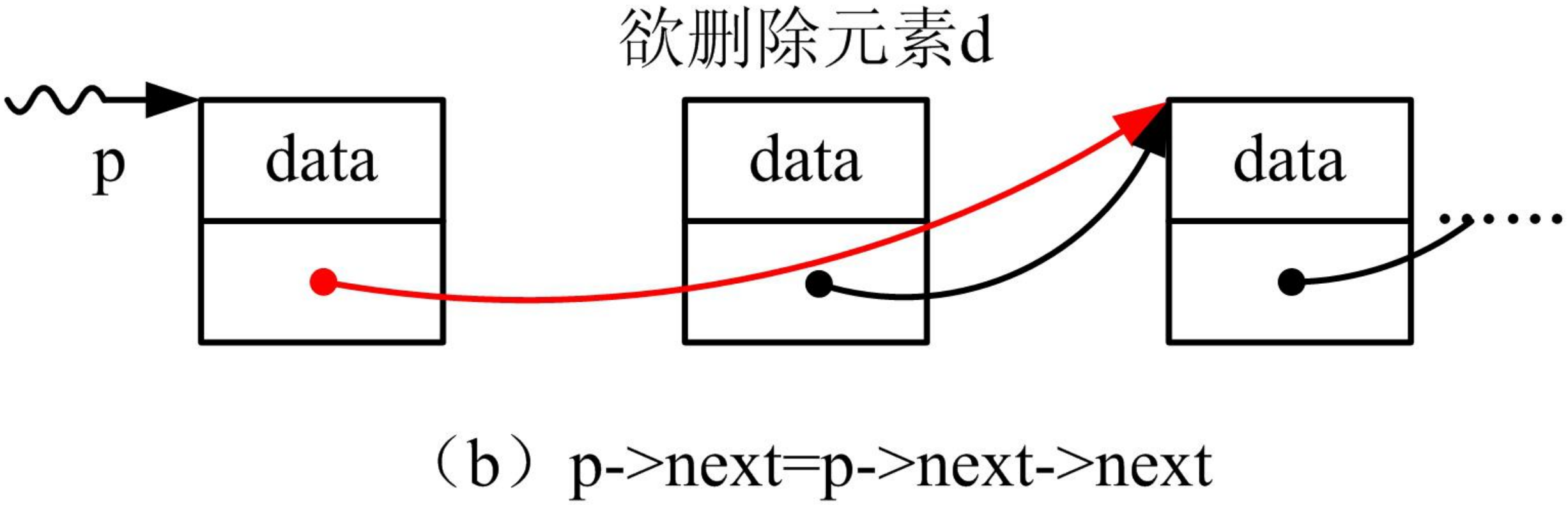
- 将链表中第i个结点删去的方法是先在单链表中找到第i-1个结点p，再删除其后的结点。

图22.5 单链表删除结点示意



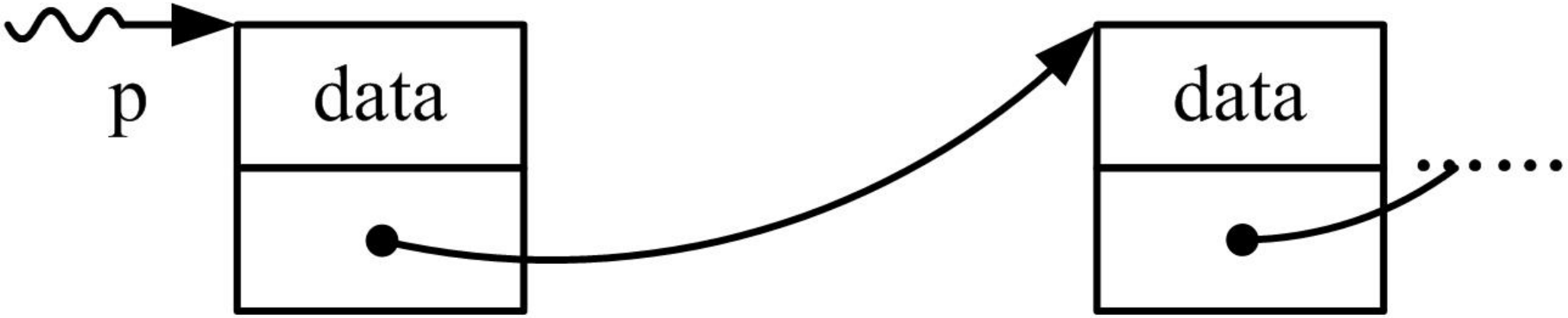
（a）删除前

图22.5 单链表删除结点示意



（b）p->next=p->next->next

图22.5 单链表删除结点示意



（c）删除后

```
1  int ListDelete(LinkList *L,int i,ElemType *ep)
2  { //删除第i个结点,并由*ep返回其值
3    LinkList p=NULL,q=*L; //q指向头结点
4    while(q!=NULL && i>=1) { //直到第i个结点
5      p=q; //p是q的前驱
6      q=q->next; //q指向直接后继结点
7      i--;
8    }
9    if(p==NULL||q==NULL) return 0;//i值不合法返回假（0）
10   p->next=q->next; //结点删除算法
11   if(ep!=NULL) *ep=q->data; //删除结点由*ep返回其值
12   delete q; //释放结点
13   return 1; //操作成功返回真（1）
14 }
```

C程序设计