西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

结点插入和删除

1、单链表插入结点

2、单链表删除结点

3、双链表插入结点

4、双链表删除结点

9.4 结点的插入与删除

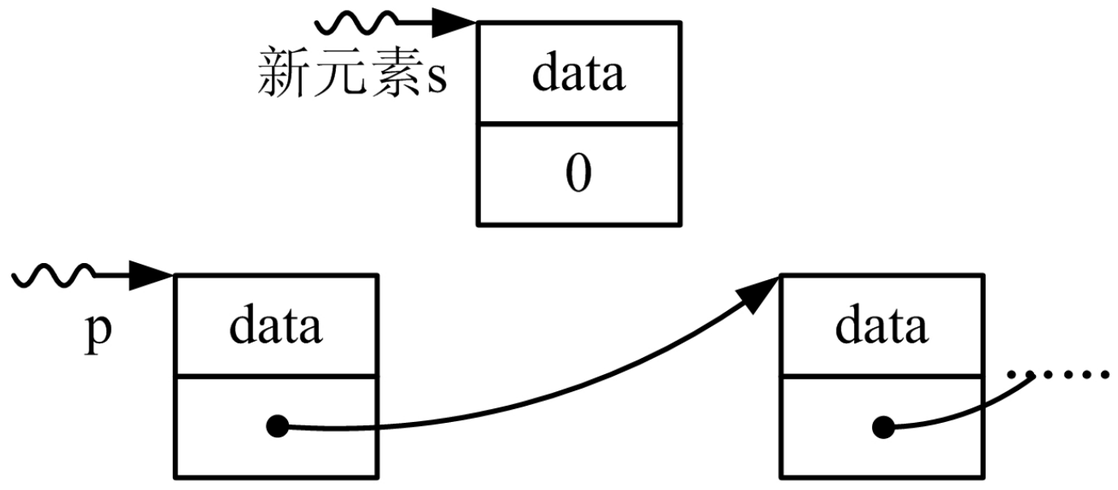
- 链表中每个结点都有指针域指向其前后结点，在进行结点插入和删除时，不能仅仅只对该结点进行操作，还要考虑其前后结点。

9.4.1 单链表插入结点

- 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。

9.4.1 单链表插入结点

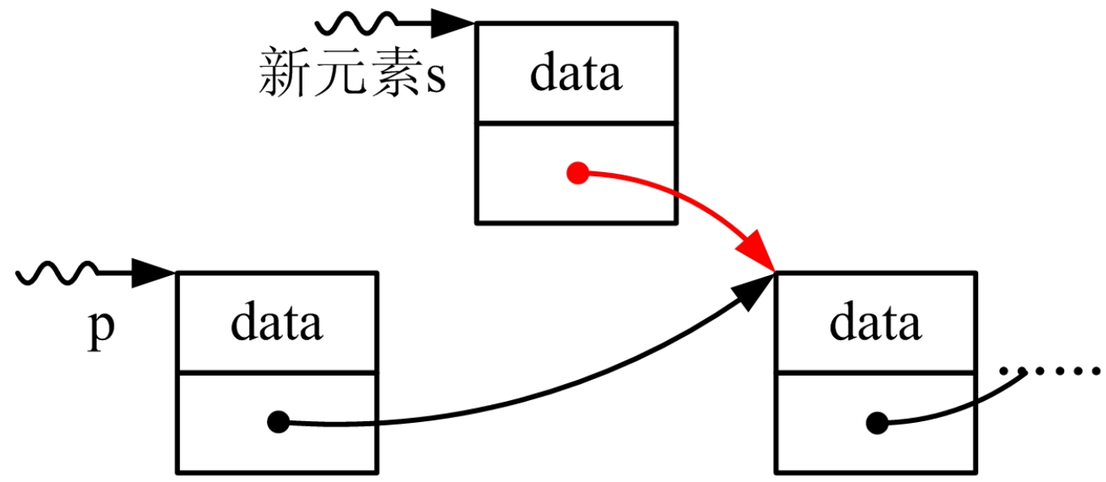
- 假设将新结点 插入到单链表的第i个结点位置上。方法是先在单链表中找到第i-1个结点p，在其后插入新结点s，如图（a）所示。



（a）插入前

9.4.1 单链表插入结点

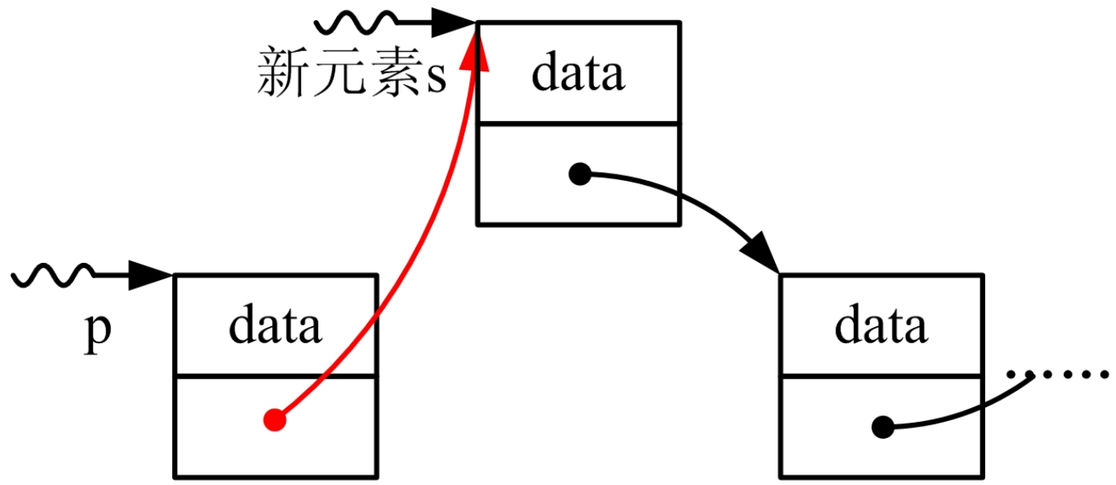
- 为了插入结点s，先让结点s的指针域指向结点p的后一个结点（即p->next），如图（b）所示；



（b）s->next=p->next

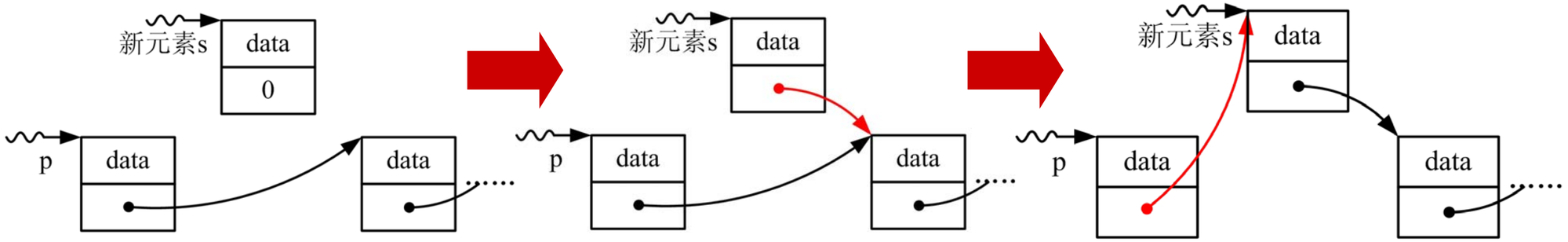
9.4.1 单链表插入结点

- 然后修改结点p的指针域，令其指向结点s，如图（c）所示，从而实现3个结点指向关系的变化。

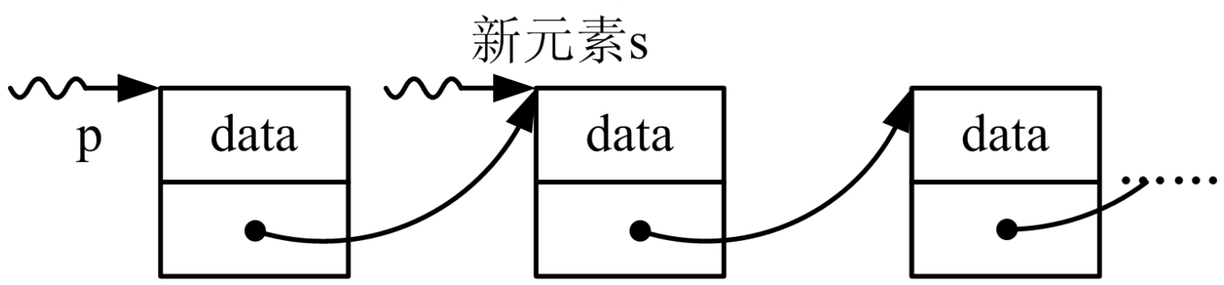


（c）p->next=s

9.4.1 单链表插入结点



（a）插入前　　（b）s->next=p->next　　（c）p->next=s



（d）插入后

9.4.1 单链表插入结点

- 实现上述步骤的C语言语句如下：

```
s->next=p->next, p->next=s; //结点插入算法
```

- 请注意，这两个表达式的顺序不能颠倒。因为若先修改结点p的指针域指向向结点s，结点p的后一个结点（p->next）就此从链表中断开，再让结点s的指针域指向结点p的后一个结点已成错误的。

9.4.1 单链表插入结点

在单链表中第i个位置上插入元素e的新结点s的算法如下：

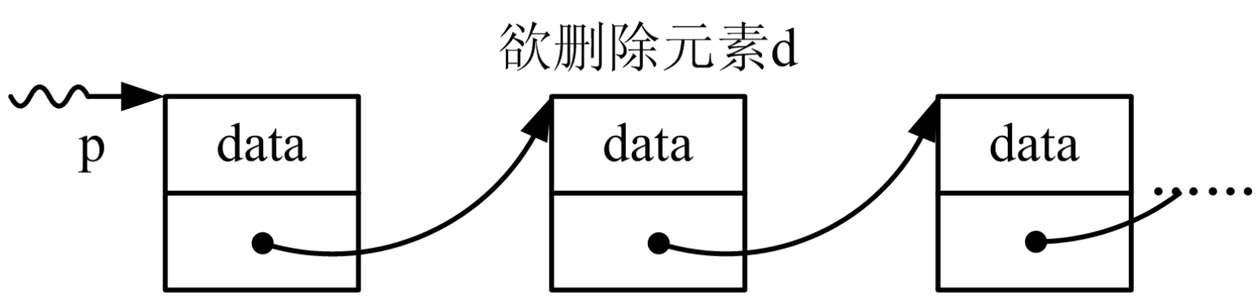
```
1  int ListInsert(LinkList *L,int i,ElemType e)
2  { //在第i个位置之前插入元素e
3    LinkList s,p=*L; //p指向头结点
4    while(p!=NULL && i>1) { //寻找第i-1个结点
5      p=p->next; //p指向直接后继结点
6      i--;
7    }
8    if(p==NULL||i<1) return 0; //i值不合法返回假（0）
9    s=(LinkList)malloc(sizeof(LNode)); //创建新结点
10   s->data=e;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

9.4.2 单链表删除结点

- 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。

9.4.2 单链表删除结点

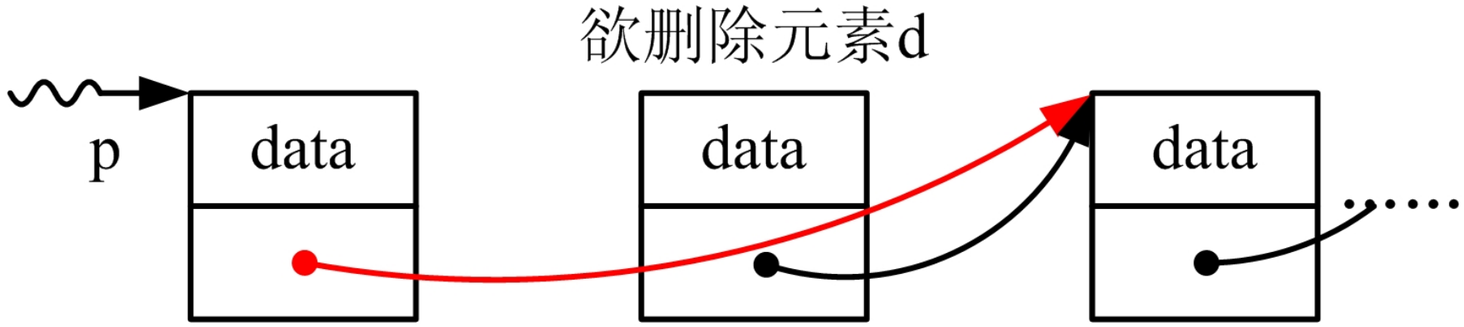
- 将链表中第i个结点删去的方法是先在单链表中找到第i-1个结点p，再删除其后的结点，如图（a）所示。



（a）删除前

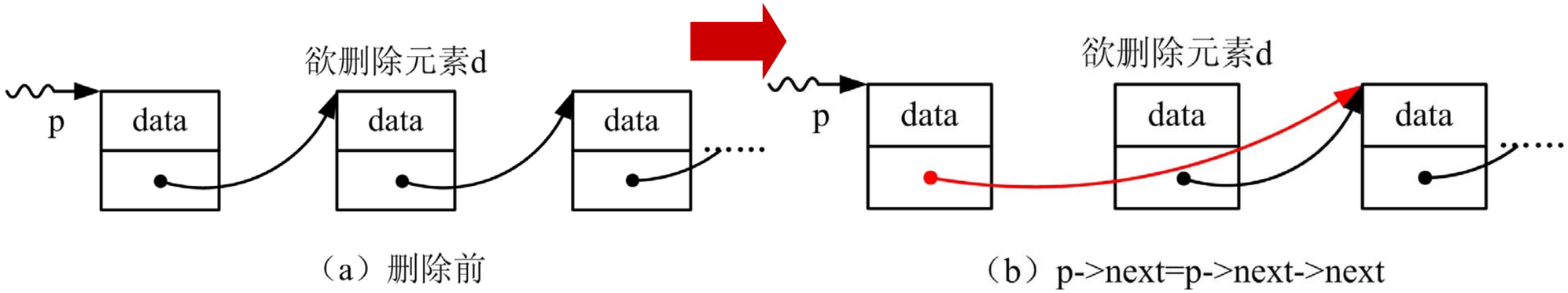
9.4.2 单链表删除结点

- 若要删除结点p的后一个结点（即p->next），只需要将p的指针域指向p->next的后一个结点（即p->next->next），如图（b）所示。

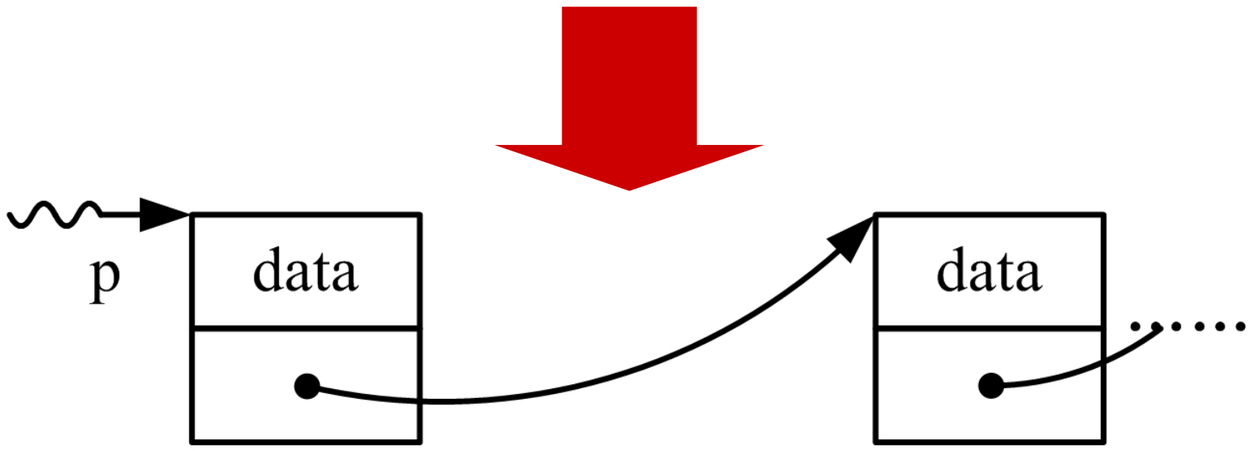


（b）p->next=p->next->next

9.4.2 单链表删除结点



（a）删除前　　（b）p->next=p->next->next



（c）删除后

9.4.2 单链表删除结点

- 实现上述步骤的C语言语句如下：

```
p->next=p->next->next; //结点删除算法
```

- 删除单链表中第i个位置结点的算法如下

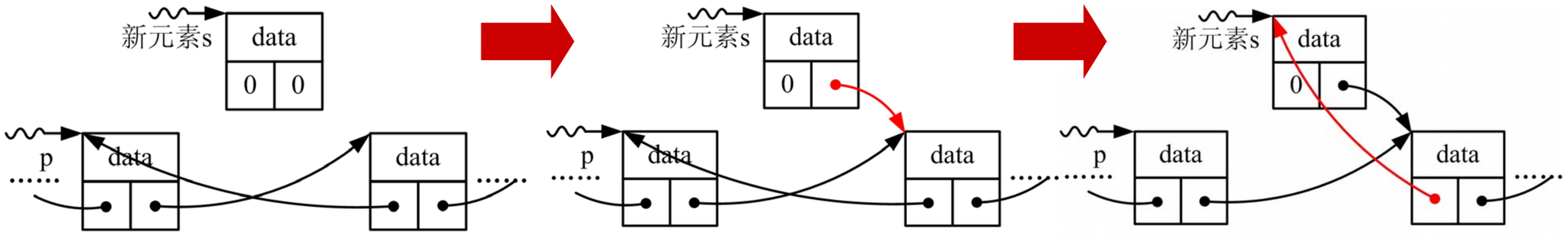
9.4.2 单链表删除结点

```
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   free(q); //释放结点
13   return 1; //操作成功返回真（1）
14 }
```

9.4.3 双链表插入结点

- 双向链表的插入结点与单向链表类似，不过需要要维护两条链的指针域变化，即前向链next和后向链prev。
- 假设在双向链表中结点p之后插入新结点s，其指针域的变化过程如图所示。

9.4.3 双链表插入结点


新元素s
data
0
0
p
data
data
（a）插入前
新元素s
data
0
p
data
data
（b）s->next=p->next
新元素s
data
0
p
data
data
（c）p->next->prev=s
新元素s
data
p
data
data
（d）s->prev=p
新元素s
data
p
data
data
（e）p->next=s
新元素s
p
data
data
data
（f）插入后

9.4.3 双链表插入结点

- 实现其步骤的C语言语句如下：

```
s->next=p->next, p->next->prev=s; //结点插入算法
s->prev=p, p->next=s;
```

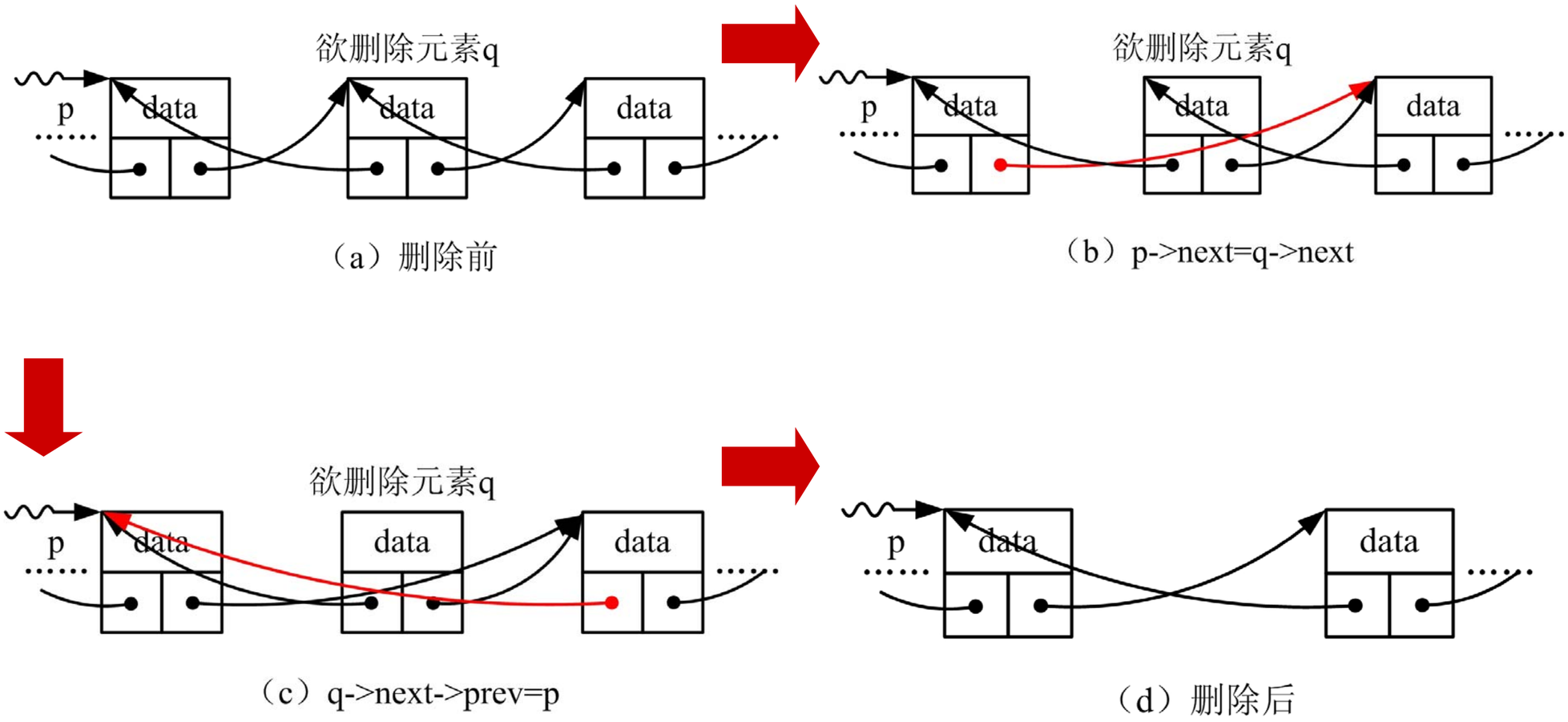
9.4.3 双链表插入结点

```
1  int ListInsert(DLinkList *L,int i,ElemType e)
2  { //在第i个位置之前插入元素e
3    DLinkList 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=(DLinkList)malloc(sizeof(DNode)); //创建新结点
10   s->data=e; //插入L中
11   s->next=p->next; //结点插入算法
12   if (p->next!=NULL) p->next->prev=s;
13   s->prev=p, p->next=s;
14   return 1; //操作成功返回真（1）
15 }
```

9.4.4 双链表删除结点

- 假设删除双向链表中结点p的后续结点，其指针域的变化过程如图所示。

9.4.4 双链表删除结点


欲删除元素q
p
data
data
data
（a）删除前
欲删除元素q
p
data
data
data
（b）p->next=q->next
欲删除元素q
p
data
data
data
（c）q->next->prev=p
p
data
data
（d）删除后

9.4.4 双链表删除结点

- 实现其步骤的C语言语句如下：

```
p->next=q->next; //结点删除算法
q->next->prev=p;
```

- 删除双链表中第i个位置结点的算法如下：

9.4.4 双链表删除结点

```
1  int ListDelete(DLinkList *L,int i,ElemType *ep)
2  { //删除第i个结点,并由*ep返回其值
3    DLinkList 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 (q->next!=NULL) q->next->prev=p;
12   if(ep!=NULL) *ep=q->data; //删除结点由*ep返回其值
13   free(q); //释放结点
14   return 1; //操作成功返回真（1）
15 }
```

9.4.4 双链表删除结点

- 单链表接口函数大全：
- 数据类型定义、头插法/尾插法创建链表、销毁链表、链表重置为空、检测链表是否为空、返回链表中元素个数、返回链表第i个元素、查找链表元素、返回链表前驱元素、返回链表后继元素、插入节点、删除节点、遍历链表

9.4.4 双链表删除结点

例9.1

```
 1 typedef int ElemType; //简单的数据元素类型
 2 typedef struct tagLNode { //单链表结点类型
 3   ElemType data; //数据域
 4   struct tagLNode *next; //指针域：指向直接后继结点
 5 } LNode, *LinkList; //LNode为单链表结构体类型，LinkList为单链表
指针类型
 6 void CreateLinkF(LinkList *L,int n,void(*input)(ElemType*))
 7 { //头插法创建单链表，调用input输入函数输入数据
 8   LinkList s;
 9   *L=(LinkList)malloc(sizeof(LNode)); //创建头结点
10   (*L)->next=NULL; //初始时为空表
11   for (; n>0; n--) { //创建n个结点链表
12     s=(LinkList)malloc(sizeof(LNode)); //创建新结点
13     input(&s->data); //调用input输入数据域
14     s->next=(*L)->next; //将s增加到开始结点之前
```

9.4.4 双链表删除结点

例9.1

```
15      (*L)->next=s; //头结点之后
16    }
17  }
18  void CreateLinkR(LinkList *L,int n,void(*input)(ElemType*))
19  { //尾插法创建单链表，调用input输入函数输入数据
20    LinkList p,s;
21    p=*L=(LinkList)malloc(sizeof(LNode)); //创建头结点
22    for (; n>0; n--) { //创建n个结点链表
23      s=(LinkList)malloc(sizeof(LNode)); //创建新结点
24      input(&s->data); //调用input输入数据域
25      p->next=s, p=s; //将s插入到当前链表末尾
26    }
27    p->next=NULL; //尾结点
28  }
29  void InitList(LinkList *L) //构造一个空的单链表L
```

9.4.4 双链表删除结点

例9.1

```
30 {
31    *L=(LinkList)malloc(sizeof(LNode)); //创建头结点，*L指向头结点
32    (*L)->next=NULL; //初始时为空表
33 }
34 void DestroyList(LinkList *L) //销毁单链表L
35 {
36    LinkList q,p=*L;//p指向头结点
37    while(p!=NULL) { //若不是链尾继续
38       q=p->next; //指向直接后继结点
39       free(p); //释放结点存储空间
40       p=q; //直接后继结点
41    }
42    *L=NULL;//置为空表
43 }
```

9.4.4 双链表删除结点

例9.1

```
44 void ClearList(LinkList *L) //将L重置为空表
45 {
46   LinkList p,q;
47   p=(*L)->next; //p指向开始结点
48   while(p!=NULL) { //若不是链尾继续
49     q=p->next; //指向直接后继结点
50     free(p); //释放结点存储空间
51     p=q; //直接后继结点
52   }
53   (*L)->next=NULL; //初始时为空表
54 }
55 int ListEmpty(LinkList L) //检测L是否为空表
56 { //若L为空表返回TRUE，否则返回FALSE
57   return L->next==NULL;
58 }
```

9.4.4 双链表删除结点

例9.1

```
59 int ListLength(LinkList L) //返回L中数据元素个数
60 {
61   int cnt=0;
62   LinkList p=L->next; //p指向开始结点
63   while(p!=NULL) { //若不是链尾继续
64     cnt++;
65     p=p->next; //指向直接后继结点
66   }
67   return cnt; //返回0表示无数据结点
68 }
69 int GetElem(LinkList L,int i,ElemType *e) //用*e返回L中第i个
数据元素
70 {
71   LinkList p=L; //p指向头结点
```

9.4.4 双链表删除结点

例9.1

```
 72    while(p!=NULL && i>0) { //顺指针向后查找直到p指向第i个元素或链
尾结束
 73      p=p->next; //指向直接后继结点
 74      i--;
 75    }
 76    if(p==NULL||p==L) return 0; //第i个元素不存在返回假（0）
 77    if (e!=NULL) *e=p->data; //用*e返回第i个元素
 78    return 1; //操作成功返回真（1）
 79  }
 80  int LocateElem(LinkList L,ElemType e,int(*compare)(ElemTyp
e*,ElemType*))
 81  { //返回L中第1个与e满足关系compare()的数据元素的位序
 82    int i=0;
 83    LinkList p=L->next; //p指向开始结点
 84    while(p!=NULL) { //若不是链尾继续
```

9.4.4 双链表删除结点

例9.1

```
 85      i++;
 86      if(compare(&(p->data),&e)) return i; //关系成立时找到指定
元素的位序
 87      p=p->next; //指向直接后继结点
 88    }
 89    return 0; //关系不存在返回0
 90 }
 91 int PriorElem(LinkList L,ElemType cur_e,ElemType *pre_e)
 92 { //用pre_e返回cur_e元素的前驱
 93   LinkList q,p=L->next; //p指向开始结点
 94   while(p!=NULL) { //若不是链尾继续
 95     q=p->next; //q为p的直接后继结点
 96     if(q!=NULL&&q->data==cur_e&&pre_e!=NULL) {
 97       *pre_e=p->data; //*pre_e返回前驱元素
 98       return 1; //操作成功返回真（1）
```

9.4.4 双链表删除结点

例9.1

```
 99      }
100      p=q; //p指向直接后继结点
101    }
102    return 0; //不存在cur_e返回假（0）
103 }
104 int NextElem(LinkList L,ElemType cur_e,ElemType *next_e)
105 { //用next_e返回cur_e元素的后继
106    LinkList p=L->next; //p指向开始结点
107    while(p!=NULL) { //若不是链尾继续
108      if(p->data==cur_e) {
109        if (p->next!=NULL&&next_e!=NULL)
110          *next_e=p->next->data; //*next_e返回后继元素
111        return 1; //操作成功返回真（1）
112      }
113      p=p->next; //p指向直接后继结点
```

9.4.4 双链表删除结点

例9.1

```
114     }
115     return 0; //不存在cur_e返回假（0）
116 }
117 int ListInsert(LinkList *L,int i,ElemType e)
118 { //在第i个位置之前插入元素e
119     LinkList s,p=*L; //p指向头结点
120     while(p!=NULL && i>1) { //寻找第i-1个结点
121       p=p->next; //p指向直接后继结点
122       i--;
123     }
124     if(p==NULL||i<1) return 0; //i值不合法返回假（0）
125     s=(LinkList)malloc(sizeof(LNode)); //创建新结点
126     s->data=e; //插入L中
127     s->next=p->next, p->next=s; //结点插入算法
128     return 1; //操作成功返回真（1）
```

9.4.4 双链表删除结点

例9.1

```
129 }
130 int ListDelete(LinkList *L,int i,ElemType *ep)
131 { //删除第i个结点,并由*ep返回其值
132   LinkList p=NULL,q=*L; //q指向头结点
133   while(q!=NULL && i>=1) { //直到第i个结点
134     p=q; //p是q的前驱
135     q=q->next; //q指向直接后继结点
136     i--;
137   }
138   if(p==NULL||q==NULL) return 0;//i值不合法返回假（0）
139   p->next=q->next; //结点删除算法
140   if (ep!=NULL) *ep=q->data; //删除结点由*ep返回其值
141   free(q); //释放结点
142   return 1; //操作成功返回真（1）
143 }
```

9.4.4 双链表删除结点

例9.1

```
144 void ListTraverse(LinkList L,void(*visit)(ElemType*))
145 { //遍历L中的每个元素且调用函数visit访问它
146   LinkList p=L->next; //p指向开始结点
147   while(p!=NULL) { //若不是链尾继续
148     visit(&(p->data)); //调用函数visit()访问结点
149     p=p->next; //p指向直接后继结点
150   }
151 }
```

9.4.4 双链表删除结点

【例9.1】

单链表接口函数调用示例。

9.4.4 双链表删除结点

例9.1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "EX0901.c"
4 void input(ElemType *ep) //实现数据域元素输入的定制函数
5 { //在函数中可以写更加复杂、任意形式、任意数目的输入
6   scanf("%d",ep);
7 }
8 void visit(ElemType *ep) //实现链表遍历时结点访问的定制函数
9 { //在函数中对结点*ep实现定制的操作，例如输出
10   printf("%d ",*ep);
11 }
12 int compare(ElemType *ep1,ElemType *ep2) //实现两个数据元素关
系比较的定制函数
13 { //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
14   if (*ep1==*ep2) return 1; //满足相等关系返回真（1）
```

9.4.4 双链表删除结点

例9.1

```
15    return 0; //不满足关系返回假（0）
16 }
17 int LastElem(LinkList L,ElemType *e) //用e返回尾结点元素
18 {
19    LinkList q=NULL, p=L; //指向头结点
20    while (p!=NULL) { //若不是链尾继续
21       q=p;
22       p=p->next; //指向直接后继结点
23    }
24    if (q!=NULL) {
25       *e=q->data;
26       return 1; //操作成功返回真（1）
27    }
28    return 0; //操作失败返回假（0）
29 }
```

9.4.4 双链表删除结点

例9.1

```
30 int LinkRing(LinkList L) //判断链表是否为循环链表
31 {
32   LinkList p=L; //指向头结点
33   while (p!=NULL) { //若是链尾结束
34     p=p->next; //指向直接后继结点
35     if (p==L) return 1; //是循环链表返回真（1）
36   }
37   return 0; //不是循环链表返回假（0）
38 }
39 void LinkContact(LinkList L1,LinkList *L2) //两个链表相连
40 {
41   LinkList q=NULL,p=L1; //p指向链表1头结点
42   while (p!=NULL) { //若是链表1链尾结束
43     q=p;
44     p=p->next; //指向直接后继结点
```

9.4.4 双链表删除结点

例9.1

```
45    }
46    if (q!=NULL && (*L2)!=NULL) {
47      q->next=(*L2)->next; //链表1尾结点指向链表2开始结点
48      free(*L2); //释放链表2头结点
49      *L2=NULL;
50    }
51  }
52  void ReverseLink(LinkList L) //链表逆序
53  {
54    LinkList p,q,bq;
55    if(L==NULL||L->next==NULL) return; //空链表或只有头结点直接
返回
56    p=L->next; //前向链：指向开始结点
57    q=L->next->next; //后向链：指向直接后继结点
58    p->next=NULL; //开始结点设为新尾结点
```

9.4.4 双链表删除结点

例9.1

```
59    while(q!=NULL) { //若是链尾结束
60      bq=q->next; //临时记录直接后继结点
61      q->next=p; //当前结点next指向前驱结点
62      p=q; //当前结点为下一个前驱结点
63      q=bq; //指向直接后继结点
64    }
65    L->next=p; //头结点next设为原尾结点
66  }
67  int main()
68  {
69    LinkList L,L1;
70    ElemType e;
71    CreateLinkF(&L,5,input); //创建单链表L
72    printf("L=");
73    ListTraverse(L,visit); //显示链表数据
```

9.4.4 双链表删除结点

例9.1

```
 74    printf("\nlength=%d\n",ListLength(L)); //链表长度
 75    LastElem(L,&e); //返回尾结点
 76    printf("last=%d\n",e); //尾结点
 77    printf("ring=%d\n",LinkRing(L)); //循环链表
 78    if (GetElem(L,1,&e)) //返回指定位置的元素
 79      printf("get=%d\n",e);
 80    printf("locate=%d\n",LocateElem(L,5,compare)); //返回满足
条件的元素
 81    PriorElem(L,3,&e); //返回指定元素的前驱
 82    printf("->3=%d\n",e);
 83    NextElem(L,3,&e); //返回指定元素的后继
 84    printf("3->=%d\n",e);
 85    ListInsert(&L,6,-1); //插入结点
 86    printf("Insert=");
 87    ListTraverse(L,visit); //显示链表数据
```

9.4.4 双链表删除结点

例9.1

```
 88    ListDelete(&L,3,NULL); //删除结点
 89    printf("\nDelete=");
 90    ListTraverse(L,visit); //显示链表数据
 91    printf("\n");
 92    CreateLinkR(&L1,5,input); //创建链表
 93    LinkContact(L,&L1);
 94    printf("Contact=");
 95    ListTraverse(L,visit); //显示链表数据
 96    ReverseLink(L); //链表逆序
 97    printf("\nReverse=");
 98    ListTraverse(L,visit); //显示链表数据
 99    printf("\nDestroy\n");
100    DestroyList(&L); //销毁链表
101    return 0;
102 }
```

9.4.4 双链表删除结点

例9.1

```
#include <stdio.h>
#include <stdlib.h>
#include "EX0901.c"
void CreateList(LinkList *L,int n)
 {  //逆位序(插在表头)输入n个元素的值，建立带表头结构的单链线性表L
  int i;
  LinkList p;
  *L=(LinkList)malloc(sizeof(LNode));
  (*L)->next=NULL; //先建立一个带头结点的单链表
  printf("请输入%d个数据\n",n);
  for(i=n;i>0;--i)
  {
    p=(LinkList)malloc(sizeof(LNode)); //生成新结点
    scanf("%d",&p->data); //输入元素值
    p->next=(*L)->next; //插入到表头
```

9.4.4 双链表删除结点

例9.1

```
16      (*L)->next=p;
17    }
18 }
19 void CreateList2(LinkList *L,int n)
20 { //正位序(插在表尾)输入n个元素的值，建立带表头结构的单链线性表
21    int i;
22    LinkList p,q;
23    *L=(LinkList)malloc(sizeof(LNode)); //生成头结点
24    (*L)->next=NULL;
25    q=*L;
26    printf("请输入%d个数据\n",n);
27    for(i=1;i<=n;i++)
28    {
29      p=(LinkList)malloc(sizeof(LNode));
30      scanf("%d",&p->data);
```

9.4.4 双链表删除结点

例9.1

```
31        q->next=p;
32        q=q->next;
33     }
34     p->next=NULL;
35  }
36  void MergeList(LinkList La,LinkList *Lb,LinkList *Lc)
37  { //已知单链线性表La和Lb的元素按值非递减排列。
38    //归并La和Lb得到新的单链线性表Lc，Lc的元素也按值非递减排列
39    LinkList pa=La->next,pb=(*Lb)->next,pc;
40    *Lc=pc=La; //用La的头结点作为Lc的头结点
41    while(pa&&pb)
42      if(pa->data<=pb->data)
43      {
44        pc->next=pa;
45        pc=pa;
```

9.4.4 双链表删除结点

例9.1

```
46          pa=pa->next;
47      }
48      else
49      {
50          pc->next=pb;
51          pc=pb;
52          pb=pb->next;
53      }
54      pc->next=pa?pa:pb; //插入剩余段
55      free(*Lb); //释放Lb的头结点
56      Lb=NULL;
57  }
58  void visit(ElemType *ep) //ListTraverse()调用的函数(类型要一致)
59  {
60    printf("%d ",*ep);
```

9.4.4 双链表删除结点

例9.1

```
61 }
62 int main()
63 {
64   int n=5;
65   LinkList La,Lb,Lc;
66   printf("按非递减顺序, ");
67   CreateList2(&La,n); //正位序输入n个元素的值
68   printf("La="); //输出链表La的内容
69   ListTraverse(La,visit);
70   printf("按非递增顺序, ");
71   CreateList(&Lb,n); //逆位序输入n个元素的值
72   printf("Lb="); //输出链表Lb的内容
73   ListTraverse(Lb,visit);
74   MergeList(La,&Lb,&Lc); //按非递减顺序归并La和Lb,得到新表Lc
75   printf("Lc="); //输出链表Lc的内容
```

9.4.4 双链表删除结点

例9.1

```
76    ListTraverse(Lc,visit);
77    return 0;
78 }
```

9.4.4 双链表删除结点

【例9.1a】

将一个链表添加到另一个链接表中。

9.4.4 双链表删除结点

例9.1

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "EX0901.c"
 4 int equal(ElemType *c1,ElemType *c2)
 5 { //判断是否相等的函数，Union()用到
 6   if(*c1==*c2)
 7     return 1;
 8   else
 9     return 0;
10 }
11 void Union(LinkList La,LinkList Lb)
12 { //将所有在线性表Lb中但不在La中的数据元素插入到La中
13   ElemType e;
14   int La_len,Lb_len;
15   int i;
```

9.4.4 双链表删除结点

例9.1

```
16    La_len=ListLength(La); //求线性表的长度
17    Lb_len=ListLength(Lb);
18    for(i=1;i<=Lb_len;i++)
19    {
20      GetElem(Lb,i,&e); //取Lb中第i个数据元素赋给e
21      if(!LocateElem(La,e,equal)) //La中不存在和e相同的元素,则插
入之
22          ListInsert(&La,++La_len,e);
23    }
24 }
25 void print(ElemType *c)
26 {
27    printf("%d ",*c);
28 }
29 int main()
```

9.4.4 双链表删除结点

例9.1

```
{
  LinkList La,Lb;
  int j;
  InitList(&La);
  for(j=1;j<=5;j++) //在表La中插入5个元素
    ListInsert(&La,j,j);
  printf("La= "); //输出表La的内容
  ListTraverse(La,print);
  InitList(&Lb); //也可不判断是否创建成功
  for(j=1;j<=5;j++) //在表Lb中插入5个元素
    ListInsert(&Lb,j,2*j);
  printf("Lb= "); //输出表Lb的内容
  ListTraverse(Lb,print);
  Union(La,Lb);
  printf("new La= "); //输出新表La的内容
```

9.4.4 双链表删除结点

例9.1

```
45    ListTraverse(La,print);
46    return 0;
47 }
```

9.4.4 双链表删除结点

【例9.1a】

归并2个链表。

9.4.4 双链表删除结点

例9.1

```
#include <stdio.h>
#include <stdlib.h>
#include "EX0901.c"
void MergeList(LinkList La,LinkList Lb,LinkList *Lc)
{ //已知线性表La和Lb中的数据元素按值非递减排列。
  //归并La和Lb得到新的线性表Lc,Lc的数据元素也按值非递减排列
  int i=1,j=1,k=0;
  int La_len,Lb_len;
  ElemType ai,bj;
  InitList(Lc); //创建空表Lc
  La_len=ListLength(La);
  Lb_len=ListLength(Lb);
  while(i<=La_len&&j<=Lb_len) //表La和表Lb均非空
  {
    GetElem(La,i,&ai);
```

9.4.4 双链表删除结点

例9.1

```
16      GetElem(Lb,j,&bj);
17      if(ai<=bj)
18      {
19        ListInsert(Lc,++k,ai);
20        ++i;
21      }
22      else
23      {
24        ListInsert(Lc,++k,bj);
25        ++j;
26      }
27    }
28    while(i<=La_len) //表La非空且表Lb空
29    {
30      GetElem(La,i++,&ai);
```

9.4.4 双链表删除结点

例9.1

```
31      ListInsert(Lc,++k,ai);
32    }
33    while(j<=Lb_len) //表Lb非空且表La空
34    {
35      GetElem(Lb,j++,&bj);
36      ListInsert(Lc,++k,bj);
37    }
38  }
39  void print(ElemType *c)
40  {
41    printf("%d ",*c);
42  }
43  int main()
44  {
45    LinkList La,Lb,Lc;
```

9.4.4 双链表删除结点

例9.1

```
46    int j,a[4]={3,5,8,11},b[7]={2,6,8,9,11,15,20};
47    InitList(&La); //创建空表La
48    for(j=1;j<=4;j++) //在表La中插入4个元素
49      ListInsert(&La,j,a[j-1]);
50    printf("La= "); //输出表La的内容
51    ListTraverse(La,print);
52    InitList(&Lb); //创建空表Lb
53    for(j=1;j<=7;j++) //在表Lb中插入7个元素
54      ListInsert(&Lb,j,b[j-1]);
55    printf("Lb= "); //输出表Lb的内容
56    ListTraverse(Lb,print);
57    MergeList(La,Lb,&Lc);
58    printf("Lc= "); //输出表Lc的内容
59    ListTraverse(Lc,print);
60    return 0;
```

9.4.4 双链表删除结点

例9.1

```
61  }
```

9.4.4 双链表删除结点

【例9.2】

双链表接口函数调用示例。

9.4.4 双链表删除结点

例9.2

```
typedef int ElemType; //数据元素的简单类型
typedef struct tagDNode { //双链表结点类型
  ElemType data; //数据域
  struct tagDNode *prev,*next; //指针域：分别指向前驱结点和直接后继结点
} DNode, *DLinkList; //DNode为双链表结构体类型，DLinkList为双链表指针类型
//（双链表）
void CreateLinkF(DLinkList *L,int n,void(*input)(ElemType *))
{ //头插法创建双链表，调用input输入函数输入数据
  DLinkList s;
  *L=(DLinkList)malloc(sizeof(DNode)); //创建头结点
  (*L)->prev=(*L)->next=NULL; //初始时为空表
  for (; n>0; n--) { //创建n个结点链表
```

9.4.4 双链表删除结点

例9.2

```
 13      s=(DLinkList)malloc(sizeof(DNode)); //创建新结点
 14      input(&s->data); //调用input输入数据域
 15      s->next=(*L)->next; //将s增加到开始结点之前
 16      if ((*L)->next!=NULL) (*L)->next->prev=s; //开始结点之前
是s
 17      (*L)->next=s , s->prev=*L; //头结点之后是s，s之前是头结点
 18    }
 19 }
 20 //（双链表）
 21 void CreateLinkR(DLinkList *L,int n,void(*input)(ElemType
*))
 22 { //尾插法创建双链表，调用input输入函数输入数据
 23   DLinkList p,s;
 24   p=*L=(DLinkList)malloc(sizeof(DNode)); //创建头结点
 25   (*L)->prev=(*L)->next=NULL; //初始时为空表
```

9.4.4 双链表删除结点

例9.2

```
26    for (; n>0; n--) { //创建n个结点链表
27      s=(DLinkList)malloc(sizeof(DNode)); //创建新结点
28      input(&s->data); //调用input输入数据域
29      s->next=NULL; //当前结点是尾结点
30      p->next=s , s->prev=p , p=s; //将s增加到链尾
31    }
32 }
33 //（双链表）
34 void InitList(DLinkList *L) //构造一个空的单链表L
35 {
36    *L=(DLinkList)malloc(sizeof(DNode)); //创建头结点，*L指向头
结点
37    (*L)->prev=(*L)->next=NULL; //初始时为空表
38 }
39 //（单链表与双链表相同）
```

9.4.4 双链表删除结点

例9.2

```
40 void DestroyList(DLinkList *L) //销毁单链表L
41 {
42   DLinkList q,p=*L;//p指向头结点
43   while(p!=NULL) { //若不是链尾继续
44     q=p->next; //指向直接后继结点
45     free(p); //释放结点存储空间
46     p=q; //直接后继结点
47   }
48   *L=NULL;//置为空表
49 }
50 //（双链表）
51 void ClearList(DLinkList *L) //将L重置为空表
52 {
53   DLinkList p,q;
54   p=(*L)->next; //p指向开始结点
```

9.4.4 双链表删除结点

例9.2

```
55    while(p!=NULL) { //若不是链尾继续
56      q=p->next; //指向直接后继结点
57      free(p); //释放结点存储空间
58      p=q; //直接后继结点
59    }
60    (*L)->prev=(*L)->next=NULL; //初始时为空表
61  }
62  //（单链表与双链表相同）
63  int ListEmpty(DLinkList L) //检测L是否为空表
64  { //若L为空表返回TRUE，否则返回FALSE
65    return L->next==NULL;
66  }
67  //（单链表与双链表相同）
68  int ListLength(DLinkList L) //返回L中数据元素个数
69  {
```

9.4.4 双链表删除结点

例9.2

```
70    int cnt=0;
71    DLinkList p=L->next; //p指向开始结点
72    while(p!=NULL) { ///若不是链尾继续
73       cnt++;
74       p=p->next; //指向直接后继结点
75    }
76    return cnt; //返回0表示无数据结点
77 }
78 //（单链表与双链表相同）
79 int GetElem(DLinkList L,int i,ElemType *e) //用e返回L中第i个
数据元素
80 {
81    DLinkList p=L; //p指向头结点
82    while(p!=NULL && i>0) { //顺指针向后查找直到p指向第i个元素或链
尾结束
```

9.4.4 双链表删除结点

例9.2

```
 83      p=p->next; //指向直接后继结点
 84      i--;
 85    }
 86    if(p==NULL||p==L) return 0; //第i个元素不存在返回假（0）
 87    if(e!=NULL) *e=p->data; //用e返回第i个元素
 88    return 1; //操作成功返回真（1）
 89  }
 90  //（单链表与双链表相同）
 91  int LocateElem(DLinkList L,ElemType e,int(*compare)(ElemTy
pe*,ElemType*))
 92  { //返回L中第1个与e满足关系compare()的数据元素的位序
 93    int i=0;
 94    DLinkList p=L->next; //p指向开始结点
 95    while(p!=NULL) { //若不是链尾继续
 96      i++;
```

9.4.4 双链表删除结点

例9.2

```
 97      if(compare(&(p->data),&e)) return i; //关系成立时找到指定
元素的位序
 98      p=p->next; //指向直接后继结点
 99    }
100    return 0; //关系不存在返回0
101 }
102 //（单链表与双链表相同）
103 int PriorElem(DLinkList L,ElemType cur_e,ElemType *pre_e)
104 { //用pre_e返回cur_e元素的前驱
105   DLinkList q,p=L->next; //p指向开始结点
106   while(p!=NULL) { //若不是链尾继续
107     q=p->next; //q为p的直接后继结点
108     if(q!=NULL&&q->data==cur_e&&pre_e!=NULL) {
109       *pre_e=p->data; //*pre_e返回前驱元素
110       return 1; //操作成功返回真（1）
```

9.4.4 双链表删除结点

例9.2

```
111       }
112       p=q; //p指向直接后继结点
113     }
114     return 0; //不存在cur_e返回假（0）
115 }
116 //（单链表与双链表相同）
117 int NextElem(DLinkList L,ElemType cur_e,ElemType *next_e)
118 { //用next_e返回cur_e元素的后继
119   DLinkList p=L->next; //p指向开始结点
120   while(p!=NULL) { //若不是链尾继续
121     if(p->data==cur_e) {
122       if (p->next!=NULL&&next_e!=NULL)
123         *next_e=p->next->data; //*next_e返回后继元素
124       return 1; //操作成功返回真（1）
125     }
```

9.4.4 双链表删除结点

例9.2

```
126       p=p->next; //p指向直接后继结点
127     }
128     return 0; //不存在cur_e返回假（0）
129 }
130 //（双链表）
131 int ListInsert(DLinkList *L,int i,ElemType e)
132 { //在第i个位置之前插入元素e
133     DLinkList s,p=*L; //p指向头结点
134     while(p!=NULL && i>1) { //寻找第i-1个结点
135       p=p->next; //p指向直接后继结点
136       i--;
137     }
138     if(p==NULL||i<1) return 0; //i值不合法返回假（0）
139     s=(DLinkList)malloc(sizeof(DNode)); //创建新结点
140     s->data=e; //插入L中
```

9.4.4 双链表删除结点

例9.2

```
141    s->next=p->next; //结点插入算法
142    if (p->next!=NULL) p->next->prev=s;
143    s->prev=p, p->next=s;
144    return 1; //操作成功返回真（1）
145 }
146 //（双链表）
147 int ListDelete(DLinkList *L,int i,ElemType *ep)
148 { //删除第i个结点,并由*ep返回其值
149    DLinkList p=NULL,q=*L; //q指向头结点
150    while(q!=NULL && i>=1) { //直到第i个结点
151       p=q; //p是q的前驱
152       q=q->next; //q指向直接后继结点
153       i--;
154    }
155    if(p==NULL||q==NULL) return 0;//i值不合法返回假（0）
```

9.4.4 双链表删除结点

例9.2

```
156    p->next=q->next; //结点删除算法
157    if (q->next!=NULL) q->next->prev=p;
158    if(ep!=NULL) *ep=q->data; //删除结点由*ep返回其值
159    free(q); //释放结点
160    return 1; //操作成功返回真（1）
161 }
162 //（单链表与双链表相同）
163 void ListTraverse(DLinkList L,void(*visit)(ElemType*))
164 { //遍历L中的每个元素且调用函数visit访问它
165    DLinkList p=L->next; //p指向开始结点
166    while(p!=NULL) { //若不是链尾继续
167       visit(&(p->data)); //调用函数visit()访问结点
168       p=p->next; //p指向直接后继结点
169    }
170 }
```

9.4.4 双链表删除结点

例9.2

```
#include <stdio.h>
#include <stdlib.h>
#include "EX0902.c"
void input(ElemType *ep) //实现数据域元素输入的定制函数
{ //在函数中可以写更加复杂、任意形式、任意数目的输入
  scanf("%d",ep);
}
void visit(ElemType *ep) //实现链表遍历时结点访问的定制函数
{ //在函数中对结点*ep实现定制的操作，例如输出
  printf("%d ",*ep);
}
int compare(ElemType *ep1,ElemType *ep2) //实现两个数据元素关系比较的定制函数
{ //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
  if (*ep1==*ep2) return 1; //满足相等关系返回真（1）
```

9.4.4 双链表删除结点

例9.2

```
15    return 0; //不满足关系返回假（0）
16 }
17 //（单链表与双链表相同）
18 int LastElem(DLinkList L,ElemType *e) //用e返回尾结点元素
19 {
20    DLinkList q=NULL, p=L; //指向头结点
21    while (p!=NULL) { //若是链尾结束
22       q=p;
23       p=p->next; //指向直接后继结点
24    }
25    if (q!=NULL) {
26       *e=q->data;
27       return 1; //操作成功返回真（1）
28    }
29    return 0; //操作失败返回假（0）
```

9.4.4 双链表删除结点

例9.2

```
30 }
31 //（单链表与双链表相同）
32 int LinkRing(DLinkList L) //判断链表是否为循环链表
33 {
34   DLinkList p=L; //指向头结点
35   while (p!=NULL) { //若是链尾结束
36     p=p->next; //指向直接后继结点
37     if (p==L) return 1; //是循环链表返回真（1）
38   }
39   return 0; //不是循环链表返回假（0）
40 }
41 void LinkContact(DLinkList L1,DLinkList *L2) //两个链表相连
42 {
43   DLinkList q=NULL,p=L1; //p指向链表1头结点
44   while (p!=NULL) { //若是链表1链尾结束
```

9.4.4 双链表删除结点

例9.2

```
45      q=p;
46      p=p->next; //指向直接后继结点
47    }
48    if (q!=NULL && (*L2)!=NULL) {
49      q->next=(*L2)->next; //链表1尾结点指向链表2开始结点
50      free(*L2); //释放链表2头结点
51      *L2=NULL;
52    }
53  }
54  void ReverseLink(DLinkList L) //链表逆序
55  {
56    DLinkList p,q,bq;
57    if(L==NULL||L->next==NULL) return; //空链表或只有头结点直接
返回
58    p=L->next; //前向链：指向开始结点
```

9.4.4 双链表删除结点

例9.2

```
59    q=L->next->next; //后向链：指向直接后继结点
60    p->next=NULL; //开始结点设为新尾结点
61    while(q!=NULL) { //若是链尾结束
62      bq=q->next; //临时记录直接后继结点
63      q->next=p; //当前结点next指向前驱结点
64      p=q; //当前结点为下一个前驱结点
65      q=bq; //指向直接后继结点
66    }
67    L->next=p; //头结点next设为原尾结点
68 }
69 void ListRTraverse(DLinkList L,void(*visit)(ElemType*))
70 { //遍历L中的每个元素且调用函数visit访问它
71    DLinkList p,q;
72    if (L==NULL||visit==NULL) return; //L为空表或函数不存在直接
返回
```

9.4.4 双链表删除结点

例9.2

```
73    q=p=L->next; //p指向开始结点
74    while(p!=NULL) { //若是链尾结束
75      q=p;
76      p=p->next; //p指向直接后继结点
77    }
78    //双链表后向链
79    while (q!=L) {
80      visit(&(q->data)); //调用函数visit()访问结点
81      q=q->prev; //q指向前驱结点
82    }
83  }
84  int main()
85  {
86    DLinkList L;
87    ElemType e;
```

9.4.4 双链表删除结点

例9.2

```
88    CreateLinkF(&L,5,input); //创建双链表L
89    printf("L=");
90    ListTraverse(L,visit); //显示链表数据
91    printf("\nR-L=");
92    ListRTraverse(L,visit); //显示链表数据
93    printf("\nlength=%d\n",ListLength(L)); //链表长度
94    LastElem(L,&e); //返回尾结点
95    printf("last=%d\n",e); //尾结点
96    printf("ring=%d\n",LinkRing(L)); //循环链表
97    if (GetElem(L,1,&e)) //返回指定位置的元素
98      printf("get=%d\n",e);
99    printf("locate=%d\n",LocateElem(L,5,compare)); //返回满足
条件的元素
100   PriorElem(L,3,&e); //返回指定元素的前驱
101   printf("->3=%d\n",e);
```

9.4.4 双链表删除结点

例9.2

```
102    NextElem(L,3,&e); //返回指定元素的后继
103    printf("3->=%d\n",e);
104    ListInsert(&L,6,-1); //插入结点
105    printf("Insert=");
106    ListTraverse(L,visit); //显示链表数据
107    printf("\nR-L=");
108    ListRTraverse(L,visit); //显示链表数据
109    ListDelete(&L,3,&e); //删除结点
110    printf("\nDelete=");
111    ListTraverse(L,visit); //显示链表数据
112    printf("\nR-L=");
113    ListRTraverse(L,visit); //显示链表数据
114    printf("\n");
115    return 0;
116 }
```


程序设计