第2章 线性表

一．选择题

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1.A | 2.B | 3.C | 4.A | 5.D | 6.D | 7.D | 8.C | 9.B | 10.B,C | 11.1I | 11.2I | 11.3E |
| 11.4B | 11.5C | 12.B | 13.C | 14.C | 15.C |  | 16.A | 17.A | 18.A | 19.D | 20.C | 21.B |
| 22.D | 23.C | 24.B | 25.B | 26.A | 27.D |  |  |  |  |  |  |  |

二．判断题

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. × | 2.√ | 3. √ | 4.× | 5.× | 6. × | 7. × | 8.× | 9.× | 10.× | 11.× | 12.× |
| 13. × | 14. √ | 15.× | 16. √ |  |  |  |  |  |  |  |  |

部分答案解释如下。

1. 头结点并不“仅起”标识作用，并且使操作统一。另外，头结点数据域可写入链表长度，或作监视哨。

4．两种存储结构各有优缺点，应根据实际情况选用，不能笼统说哪一个好。

7．集合中元素无逻辑关系。

9．非空线性表第一个元素无前驱，最后一个元素无后继。

13．线性表是逻辑结构，可以顺序存储，也可链式存储。

三．填空题

1．顺序 2．（n-1）/2 3．py->next=px->next; px->next=py

4 ．n-i+1

5．主要是使插入和删除等操作统一，在第一个元素之前插入元素和删除第一个结点不必另作判断。另外，不论链表是否为空，链表指针不变。

6．O(1)，O(n) 7．单链表，多重链表，（动态）链表，静态链表

8．f->next=p->next; f->prior=p; p->next->prior=f; p->next=f;

9．p^.prior s^.prior^.next

10． 指针 11．物理上相邻 指针 12．4 2

13．从任一结点出发都可访问到链表中每一个元素。

14．u=p->next; p->next=u->next; free(u); 15．L->next->next==L 16．p->next!=null

17．L->next==L && L->prior==L 18．s->next=p->next;p->next=s;

19．(1) **IF** pa=NIL **THEN** **return**(true);

(2) pb<>NIL **AND** pa^.data>=pb^.data

(3) **return**(inclusion(pa,pb));

(4) pb:=pb^.next;

(5) **return**(false);

非递归算法：

(1)pre:=pb; (2) pa<>NIL AND pb<>NIL AND pb^.data>=pa^.data (3)pa:=pa^.next; pb:=pb->next;

(4)pb:=pre^.next;pre:=pb;pa:=pa^.next;(5)**IF** pa=NIL **THEN** **return**(true) **ELSE** **return**(false);

[注]：本题是在链表上求模式匹配问题。非递归算法中用指针pre指向主串中开始结点（初始时为第一元素结点）。若主串与子串对应数据相等，两串工作指针pa和pb后移；否则，主串工作指针从pre的下一结点开始（这时pre又指向新的开始结点），子串工作指针从子串第一元素开始，比较一直继续到循环条件失败。若pa为空，则匹配成功，返回true，否则，返回false。

20．A．**VAR** head:ptr B. new(p) C. p^.data:=k D. q^.next:=p E. q:=p(带头结点)

21．（1）new(h);∥生成头结点，以便于操作。

（2）r^.next:=p; (3) r^.next:=q; (4) **IF** (q=NIL) **THEN** r^.next:=p;

22．A: r^.link^.data<>max **AND** q^.link^.data<>max

B: r:=r^.link C: q^.link D: q^.link E: r^.link F: r^.link

G: r:=s（或r:= r^.link） H: r:=r^.link I: q^.link:=s^.link

23．(1)la (2)0 (3)j<i-1 (4)p↑.next (5)i<1

24.(1)head^.left:=s ∥head的前驱指针指向插入结点

(2)j:=1;

(3)p:=p^.right ∥工作指针后移

(4)s^.left:=p

(5)p^.right^.left:=s; ∥p后继的前驱是s

(6)s^.left:=p;

25.(1)i<=L.last ∥L.last 为元素个数

(2)j:=j+1 ∥有值不相等的元素

(3)L.elem[j]:=L.elem[i] ∥元素前移

(4)L.last:=j ∥元素个数

26.(A)p^.link:=q;∥拉上链，前驱指向后继

(B)p:=q;∥新的前驱

(C)p^.link:=head;∥形成循环链表

(D)j:=0; ∥计数器，记被删结点

(E)q:=p^.link∥记下被删结点

(F)p^.link=q^.link ∥删除结点

27. (1)p:=r;∥r指向工作指针s的前驱，p指向最小值的前驱。

(2)q:=s;∥q指向最小值结点，s是工作指针

(3)s:=s^.link∥工作指针后移

(4)head:=head^.next;∥第一个结点值最小；

(5)p^link:=q^.link;∥跨过被删结点（即删除一结点）

28．(1) l^.key:=x;∥头结点l这时起监视哨作用

(2) l^.freq:=p^.freq ∥头结点起监视哨作用

(3) q->pre->next=q->next; q->next->pre=q->pre; ∥先将q结点从链表上摘下

q^.next:=p; q^.pre:=p^.pre; p^.pre->next:=q; p^.pre:=q; ∥结点q插入结点p前

(4) q^.freq=0 ∥链表中无值为x的结点，将新建结点插入到链表最后（头结点前）。

29．(1)a^.key:=’@’∥a的头结点用作监视哨，取不同于a链表中其它数据域的值

(2)b^.key:=p^.key∥b的头结点起监视哨作用

(3)p:=p^.next∥找到a,b表中共同字母，a表指针后移

(4)0(m\*n)

30. C 部分：(1)p!=null ∥链表未到尾就一直作

(2)q ∥将当前结点作为头结点后的第一元素结点插入

31. (1)L=L->next; ∥暂存后继

(2)q=L; ∥待逆置结点

(3)L=p; ∥头指针仍为L

32. (1) p^.next<>p0 (2)r:= p^.next (3) p^.next:= q0;

(4) q0:= p; (5) p:=r

33. (1)r (2)NIL (3)x<head^.data (4)p^.data<x

(5)p:=p^.next (6)p^.data>=x; (7)r (8)p

(9)r (10)NIL (11)NIL

34．(1)pa!=ha ∥或pa->exp!=-1

(2)pa->exp==0 ∥若指数为0，即本项为常数项

(3)q->next=pa->next ∥删常数项

(4)q->next ∥取下一元素

(5)=pa->coef\*pa->exp

(6)-- ∥指数项减1

(7)pa ∥前驱后移,或q->next

(8)pa->next ∥取下一元素

35．(1)q:=p; ∥q是工作指针p的前驱

(2)p^.data>m ∥p是工作指针

(3)r:=q; ∥r 记最大值的前驱，

(4)q:=p; ∥或q:=q^.next;

(5)r^.next:=q^.next; ∥或r^.next:=r^.next^.next 删最大值结点

36．(1)L->next=null ∥置空链表,然后将原链表结点逐个插入到有序表中

(2)p!=null ∥当链表尚未到尾，p为工作指针

(3)q!=null ∥查p结点在链表中的插入位置，这时q是工作指针。

(4)p->next=r->next ∥将p结点链入链表中

(5)r->next=p ∥r是q的前驱，u是下个待插入结点的指针。

37．程序（a） PASCAL部分(编者略）

程序（b） C部分

(1)(A!=null && B!=null) ∥两均未空时循环

(2)A->element==B->element ∥两表中相等元素不作结果元素

(3)B=B->link ∥向后移动B表指针

(4)A!=null ∥将A 表剩余部分放入结果表中

(5)last->link=null ∥置链表尾

1. 应用题

1．（1）选链式存储结构。它可动态申请内存空间，不受表长度（即表中元素个数）的影响，插入、删

除时间复杂度为O（1）。

（2）选顺序存储结构。顺序表可以随机存取，时间复杂度为O（1）。

2．链式存储结构一般说克服了顺序存储结构的三个弱点。首先，插入、删除不需移动元素，只修改指针，时间复杂度为O(1)；其次，不需要预先分配空间，可根据需要动态申请空间；其三，表容量只受可用内存空间的限制。其缺点是因为指针增加了空间开销，当空间不允许时，就不能克服顺序存储的缺点。

3．采用链式存储结构，它根据实际需要申请内存空间，而当不需要时又可将不用结点空间返还给系统。在链式存储结构中插入和删除操作不需要移动元素。

4．线性表 栈 队列 串 顺序存储结构和链式存储结构。

顺序存储结构的定义是：

**CONST** maxlen=线性表可能达到的最大长度；

**TYPE** sqlisttp=**RECORD**

elem:**ARRAY**[1..maxlen] **OF** ElemType;

last:0..maxlen;

**END**;

链式存储结构的定义是：

**TYPE** pointer=↑nodetype;

nodetype=**RECORD**

data:ElemType;

next:pointer;

**END**;

linklisttp=pointer;

5.顺序映射时，ai与ai+1的物理位置相邻；链表表示时ai与ai+1的物理位置不要求相邻。

6．在线性表的链式存储结构中，头指针指链表的指针，若链表有头结点则是链表的头结点的指针，头指针具有标识作用，故常用头指针冠以链表的名字。头结点是为了操作的统一、方便而设立的，放在第一元素结点之前，其数据域一般无意义（当然有些情况下也可存放链表的长度、用做监视哨等等），有头结点后，对在第一元素结点前插入结点和删除第一结点，其操作与对其它结点的操作统一了。而且无论链表是否为空，头指针均不为空。首元结点也就是第一元素结点，它是头结点后边的第一个结点。

7．见上题6。

8．(1)将next域变为两个域: pre和next，其值域均为0..maxsize。初始化时，头结点（下标为0的元素）其next域值为1，其pre域值为n（设n是元素个数，且n<maxsize）

(2) stalist[stalist[p].pre].pre;

(3) stalist[p].next;

9. 在单链表中不能从当前结点（若当前结点不是第一结点）出发访问到任何一个结点，链表只能从头指针开始，访问到链表中每个结点。在双链表中求前驱和后继都容易，从当前结点向前到第一结点，向后到最后结点，可以访问到任何一个结点。

10．本题是链表的逆置问题。设该链表带头结点，将头结点摘下，并将其指针域置空。然后从第一元素结点开始，直到最后一个结点为止，依次前插入头结点的后面，则实现了链表的逆置。

11．该算法的功能是判断链表L是否是非递减有序，若是则返回“true”；否则返回“false“。pre指向当前结点，p指向pre的后继。

12．q=p->next; p->next=q->next; free(q);

13. 设单链表的头结点的头指针为head,且pre=head；

**while**(pre->next!=p) pre=pre->next;

s->next=p; pre->next=s;

14.设单链表带头结点，工作指针p初始化为p=H->next;

(1) **while**(p!=null && p->data!=X) p=p->next;

**if**(p= =null) **return**(null);∥查找失败

**else** **return**(p);∥查找成功

(2) **while**(p!=null && p->data<X ) p=p->next;

**if**(p==null || p->data>X) **return**(null);∥查找失败

**else return**(p);

(3) **while**(p!=null && p->data>X) p=p->next;

**if**(p==null || p->data<X) **return**(null); ∥查找失败

**else return**(p); ∥查找成功

15.本程序段功能是将pa和pb链表中的值相同的结点保留在pa链表中（pa中与pb中不同结点删除），pa是结果链表的头指针。链表中结点值与从前逆序。S1记结果链表中结点个数（即pa与pb中相等的元素个数）。S2记原pa链表中删除的结点个数。

16．设 q:=p^.llink; 则

q^.rlink:=p^.rlink; p^.rlink^.llink:=q; p^.llink:=q^.llink;

q^.llink^.rlink:=p; p^.rlink:=q; q^.llink:=p

17.(1)前两个语句改为：

p.llink^.rlink<- p^.rlink;

p^.rlink^.llink<- p^.llink;

(2)后三个语句序列应改为：

q^.rlink<- p^.rlink;∥以下三句的顺序不能变

p^.rlink^.llink<- q;

p^.rlink<- q;

18．mp是一个过程，其内嵌套有过程subp。

subp(s,q)的作用是构造从s到q的循环链表。

subp(pa,pb)调用结果是将pa到pb的前驱构造为循环链表。

subp(pb,pa)调用结果是将pb到pa的前驱（指在L链表中，并非刚构造的pa循环链表中）构造为循环链表。

总之，两次调用将L循环链表分解为两个。第一个循环链表包含从pa到pb的前驱,L中除刚构造的pa到pb前驱外的结点形成第二个循环链表。

19．在指针p所指结点前插入结点s的语句如下：

s->pre=p->pre; s->next=p; p->pre->next=s; p->pre=s;

20．(A) f1<>NIL并且f2<>NIL

(B) f1↑.data < f2↑.data

(C) f2↑.data<f1↑.data

(D) f3↑.data<f1↑.data

(E) f1<- f1↑.link 或f2=f2↑.link;

21. 1）本算法功能是将双向循环链表结点的数据域按值自小到大排序，成为非递减（可能包括数据域值相等的结点）有序双向循环链表。

2）(1)r->prior=q->prior;∥将q结点摘下，以便插入到适当位置。

(2)p->next->prior=q;∥（2）（3）将q结点插入

(3)p->next=q;

(4)r=r->next;或r=q->next;∥后移指针，再将新结点插入到适当位置。

1. 算法设计题

1．[题目分析]因为两链表已按元素值递增次序排列，将其合并时，均从第一个结点起进行比较，将小的链入链表中，同时后移链表工作指针。该问题要求结果链表按元素值递减次序排列。故在合并的同时，将链表结点逆置。

LinkedList Union(LinkedListla,lb)

∥la,lb分别是带头结点的两个单链表的头指针，链表中的元素值按递增序排列，本算法将两链表合并成一个按元素值递减次序排列的单链表。

{ pa=la->next; pb=lb->next;∥pa，pb分别是链表la和lb的工作指针

la->next=null; ∥la作结果链表的头指针，先将结果链表初始化为空。

**while**(pa!=null && pb!=null) ∥当两链表均不为空时作

**if**(pa->data<=pb->data)

{ r=pa->next; ∥将pa 的后继结点暂存于r。

pa->next=la->next; ∥将pa结点链于结果表中，同时逆置。

la->next=pa;

pa=r; ∥恢复pa为当前待比较结点。

}

**else**

{r=pb->next;∥ 将pb 的后继结点暂存于r。

pb->next=la->next; ∥将pb结点链于结果表中，同时逆置。

la->next=pb;

pb=r; ∥恢复pb为当前待比较结点。

}

**while**(pa!=null) ∥将la表的剩余部分链入结果表，并逆置。

{r=pa->next; pa->next=la->next; la->next=pa; pa=r; }

**while**(pb!=null)

{r=pb->next; pb->next=la->next; la->next=pb; pb=r; }

}∥算法Union结束。

［算法讨论］上面两链表均不为空的表达式也可简写为**while**(pa&&pb),两递增有序表合并成递减有序表时，上述算法是边合并边逆置。也可先合并完，再作链表逆置。后者不如前者优化。算法中最后两个**while**语句，不可能执行两个，只能二者取一，即哪个表尚未到尾，就将其逆置到结果表中，即将剩余结点依次前插入到结果表的头结点后面。

　 与本题类似的其它题解答如下：

　（１）［问题分析］与上题类似，不同之处在于：一是链表无头结点，为处理方便，给加上头结点，处理结束再删除之；二是数据相同的结点，不合并到结果链表中；三是hb链表不能被破坏，即将hb的结点合并到结果链表时，要生成新结点。

　LinkedList Union(LinkedList ha, hb)

∥ha和hb是两个无头结点的数据域值递增有序的单链表，本算法将hb中并不出现在ha中的数据合并到ha中，合并中不能破坏hb链表。

　｛LinkedList la;

la=(LinkedList)malloc(**sizeof**(LNode));

la->next=ha;∥申请头结点，以便操作。

pa=ha; ∥pa是ha链表的工作指针

pb=hb; ∥pb是hb链表的工作指针

pre=la; ∥pre指向当前待合并结点的前驱。

**while**(pa&&pb)

**if**(pa->data<pb->data)∥处理ha中数据

{pre->next=pa;pre=pa;pa=pa->next;}

**else** **if**(pa->data>pb->data)∥处理hb中数据。

{r=(LinkedList)malloc(**sizeof**(LNode));∥申请空间

r->data=pb->data; pre->next=r;

pre=r;∥将新结点链入结果链表。

pb=pb->next;∥hb链表中工作指针后移。

}

**else**∥处理pa->data=pb->data;

{pre->next=pa; pre=pa;

pa=pa->next;∥两结点数据相等时，只将ha的数据链入。

pb=pb->next;∥不要hb的相等数据

}

**if**(pa!=null)pre->next=pa;∥将两链表中剩余部分链入结果链表。

**else** pre->next=pb;

free(la);∥释放头结点.ha,hb指针未被破坏。

}∥算法nion结束。

(2)本题与上面两题类似，要求结果指针为lc，其核心语句段如下：

pa=la->next;pb=hb->next;

lc=(LinkedList )malloc(**sizeof**(LNode));

pc=lc;∥pc是结果链表中当前结点的前驱

**while**(pa&&pb)

**if**(pa->data<pb->data)

{pc->next=pa;pc=pa;pa=pa->next;}

**else** {pc->next=pb;pc=pb;pb=pb->next;}

**if**(pa)pc->next=pa; **else** pc->next=pb;

free(la);free(lb);∥释放原来两链表的头结点。

算法时间复杂度为O（m+n），其中m和n分别为链表la和lb的长度。

2．[题目分析]本组题有6个，本质上都是链表的合并操作，合并中有各种条件。与前组题不同的是，叙述上是用线性表代表集合，而操作则是求集合的并、交、差（A∪B，A∩B，A-B）等。

本题与上面1．（2）基本相同，不同之处1．（2）中链表是“非递减有序”，（可能包含相等元素），本题是元素“递增有序”（不准有相同元素）。因此两表中合并时，如有元素值相等元素，则应删掉一个。

LinkedList Union(LinkedList ha,hb)

∥线性表A和B代表两个集合，以链式存储结构存储，元素递增有序。ha和hb分别是其链表的头指针。本算法求A和B的并集A∪B，仍用线性表表示，结果链表元素也是递增有序。

{ pa=ha->next;pb=hb->next;∥设工作指针pa和pb。

pc=ha;∥pc为结果链表当前结点的前驱指针。

**while**(pa&&pb)

**if**(pa->data<pb->data)

{pc->next=pa;pc=pa;pa=pa->next;}

**else** **if**(pa->data>pb->data)

{pc->next=pb;pc=pb;pb=pb->next;}

**else**∥处理pa->data=pb->data.

{pc->next=pa;pc=pa;pa=pa->next;

u=pb;pb=pb->next;free(u);}

**if**(pa) pc->next=pa;∥ 若ha表未空，则链入结果表。

**else** pc->next=pb;∥若hb表未空，则链入结果表。

free(hb); ∥释放hb头结点

**return**(ha);

}∥算法Union结束。

与本题类似的其它几个题解答如下：

(1) 解答完全同上2。

(2) 本题是求交集，即只有同时出现在两集合中的元素才出现在结果表中。其核心语句段如下：

pa=la->next;pb=lb->next;∥设工作指针pa和pb；

pc=la;∥结果表中当前合并结点的前驱的指针。

**while**(pa&&pb)

**if**(pa->data==pb->data)∥交集并入结果表中。

{ pc->next=pa;pc=pa;pa=pa->next;

u=pb;pb=pb->next;free(u);}

**else** **if**(pa->data<pb->data) {u=pa;pa=pa->next;free(u);}

**else** {u=pb; pb=pb->next; free(u);}

**while**(pa){ u=pa; pa=pa->next; free(u);}∥ 释放结点空间

**while**(pb) {u=pb; pb=pb->next; free(u);}∥释放结点空间

pc->next=null;∥置链表尾标记。

free(lb); ∥注： 本算法中也可对B表不作释放空间的处理

（3）本题基本与（2）相同，但要求无重复元素，故在算法中，待合并结点数据要与其前驱比较，只有在与前驱数据不同时才并入链表。其核心语句段如下。

pa=L1->next;pb=L2->next;∥pa、pb是两链表的工作指针。

pc=L1;∥L1作结果链表的头指针。

**while**(pa&&pb)

**if**(pa->data<pb->data) {u=pa;pa=pa->next;free(u);}∥删除L1表多余元素

**else** **if** (pa->data>pb->data) pb=pb->next; ∥pb指针后移

**else** ∥处理交集元素

{**if**(pc==L1) {pc->next=pa;pc=pa;pa=pa->next;} ∥处理第一个相等的元素。

**else** **if**(pc->data==pa->data){ u=pa;pa=pa->next;free(u);} ∥重复元素不进入L1表。

**else**{ pc->next=pa;pc=pa;pa=pa->next;} ∥交集元素并入结果表。

} ∥**while**

**while**(pa) {u=pa;pa=pa->next;free(u);} ∥ 删L1表剩余元素

pc->next=null; ∥置结果链表尾。

注： 本算法中对L2表未作释放空间的处理。

（4） 本题与上面（3）算法相同，只是结果表要另辟空间。

（5） [题目分析]本题首先求B和C的交集，即求B和C中共有元素，再与A求并集，同时删除重复元素，以保持结果A递增。

LinkedList union(LinkedList A,B,C)

∥A,B和C均是带头结点的递增有序的单链表，本算法实现A= A∪（B∩C），使求解结构保持递增有序。

{pa=A->next;pb=B->next;pc=C->next;∥设置三个工作指针。

pre=A; ∥pre指向结果链表中当前待合并结点的前驱。

**if**(pa->data<pb->data||pa->data<pc->data)∥A中第一个元素为结果表的第一元素。

{pre->next=pa;pre=pa;pa=pa->next;}

**else**{**while**(pb&&pc) ∥找B表和C表中第一个公共元素。

**if**(pb->data<pc->data)pb=pb->next;

**else** **if**(pb->data>pc->data)pc=pc->next;

**else** **break**; ∥找到B表和C表的共同元素就退出**while**循环。

**if**(pb&&pc)∥ 因共同元素而非B表或C表空而退出上面**while**循环。

**if**(pa->data>pb->data)∥A表当前元素值大于B表和C表的公共元素，先将B表元素链入。

{pre->next=pb;pre=pb;pb=pb->next;pc=pc->next;}∥ B，C公共元素为结果表第一元素。

}∥结束了结果表中第一元素的确定

**while**(pa&&pb&&pc)

{**while**(pb&&pc)

**if**(pb->data<pc->data) pb=pb->next;

**else** **if**(pb->data>pc->data) pc=pc->next;

**else** **break**; ∥B表和C表有公共元素。

**if**(pb&&pc)

{**while**(pa&&pa->data<pb->data) ∥先将A中小于B，C公共元素部分链入。

{pre->next=pa;pre=pa;pa=pa->next;}

**if**(pre->data!=pb->data){pre->next=pb;pre=pb;pb=pb->next;pc=pc->next;}

**else**{pb=pb->next;pc=pc->next;}∥ 若A中已有B，C公共元素，则不再存入结果表。

}

}∥ **while**(pa&&pb&&pc)

**if**(pa) pre->next=pa; ∥当B，C无公共元素（即一个表已空），将A中剩余链入。

}∥算法Union结束

[算法讨论]本算法先找结果链表的第一个元素，这是因为题目要求结果表要递增有序（即删除重复元素）。这就要求当前待合并到结果表的元素要与其前驱比较。由于初始pre=A（头结点的头指针），这时的data域无意义，不能与后继比较元素大小，因此就需要确定第一个元素。当然，不要这样作，而直接进入下面循环也可以，但在链入结点时，必须先判断pre是否等于A，这占用了过多的时间。因此先将第一结点链入是可取的。

算法中的第二个问题是要求时间复杂度为O（|A|+|B|+|C|）。这就要求各个表的工作指针只能后移（即不能每次都从头指针开始查找）。本算法满足这一要求。

最后一个问题是，当B，C有一表为空（即B和C已无公共元素时），要将A的剩余部分链入结果表。

3．[题目分析]循环单链表L1和L2数据结点个数分别为m和n ，将二者合成一个循环单链表时，需要将一个循环链表的结点（从第一元素结点到最后一个结点）插入到另一循环链表的第一元素结点前即可。题目要求“用最快速度将两表合并“，因此应找结点个数少的链表查其尾结点。

LinkedList Union(LinkedList L1,L2;**int** m,n)

∥L1和L2分别是两循环单链表的头结点的指针，m和n分别是L1和L2的长度。

∥本算法用最快速度将L1和L2合并成一个循环单链表。

{**if**(m<0||n<0) {printf(“表长输入错误\n”)；exit(0);}

**if**(m<n) ∥若m<n，则查L1循环单链表的最后一个结点。

{**if**(m==0)**return**(L2);∥L1为空表。

**else**{p=L1;

**while**(p->next!=L1) p=p->next;∥查最后一个元素结点。

p->next=L2->next;∥将L1循环单链表的元素结点插入到L2的第一元素结点前。

L2->next=L1->next;

free(L1);∥释放无用头结点。

}

}∥处理完m<n情况

**else**∥ 下面处理L2长度小于等于L1的情况

{**if**(n==0)**return**(L1);∥L2为空表。

**else**{p=L2;

**while**(p->next!=L2) p=p->next;∥查最后元素结点。

p->next=L1->next;∥将L2的元素结点插入到L1循环单链表的第一元素结点前。

L1->next=L2->next;

free(L2);∥释放无用头结点。

}

}∥算法结束。

类似本题叙述的其它题解答如下：

（1）[题目分析]本题将线性表la和lb连接，要求时间复杂度为O（1），且占用辅助空间尽量小。应该使用只设尾指针的单循环链表。

LinkedList Union(LinkedList la,lb)

∥la和lb是两个无头结点的循环单链表的尾指针，本算法将lb接在la后，成为一个单循环链表。

{ q=la->next; ∥q指向la的第一个元素结点。

la->next=lb->next; ∥将lb的最后元素结点接到lb的第一元素。

lb->next=q; ∥将lb指向la的第一元素结点，实现了lb接在la后。

**return**(lb); ∥返回结果单循环链表的尾指针lb。

}∥算法结束。

[算法讨论]若循环单链表带有头结点，则相应算法片段如下：

q=lb->next; ∥q指向lb的头结点；

lb->next=la->next; ∥lb的后继结点为la的头结点。

la->next=q->next; ∥la的后继结点为lb的第一元素结点。

free(q); ∥释放lb的头结点

**return**(lb); ∥返回结果单循环链表的尾指针lb。

（2）[题目分析]本题要求将单向链表ha和单向循环链表hb合并成一个单向链表，要求算法所需时间与链表长度无关，只有使用带尾指针的循环单链表，这样最容易找到链表的首、尾结点，将该结点序列插入到单向链表第一元素之前即可。

其核心算法片段如下（设两链表均有头结点）

q=hb->next; ∥单向循环链表的表头指针

hb->next=ha->next; ∥将循环单链表最后元素结点接在ha第一元素前。

ha->next=q->next; ∥将指向原单链表第一元素的指针指向循环单链表第一结点

free(q); ∥释放循环链表头结点。

若两链表均不带头结点，则算法片段如下：

q=hb->next; ∥q指向hb首元结点。

hb->next=ha; ∥hb尾结点的后继是ha第一元素结点。

ha=q; ∥头指针指向hb的首元结点。

4．[题目分析]顺序存储结构的线性表的插入，其时间复杂度为O（n），平均移动近一半的元素。线性表LA和LB合并时，若从第一个元素开始，一定会造成元素后移，这不符合本题“高效算法”的要求。另外，题中叙述“线性表空间足够大”也暗示出另外合并方式，即应从线性表的最后一个元素开始比较，大者放到最终位置上。设两线性表的长度各为m和n ，则结果表的最后一个元素应在m+n位置上。这样从后向前，直到第一个元素为止。

**PROC** Union(VAR LA:SeqList;LB:SeqList)

∥LA和LB是顺序存储的非递减有序线性表，本算法将LB合并到LA中，元素仍非递减有序。

m:=LA.last;n:=LB.last;∥m，n分别为线性表LA和LB的长度。

k:=m+n; ∥k为结果线性表的工作指针（下标）。

i:=m;j:=n; ∥i，j分别为线性表LA和LB的工作指针（下标）。

**WHILE**(i>0)AND(j>0)**DO**

**IF** LA.elem[i]>=LB.elem[j]

**THEN**[LA.elem[k]:=LA.elem[i];k:=k-1;i:=i-1;]

**ELSE**[LA.elem[k]:=LB.elem[j];k:=k-1;j:=j-1;]

**WHILE**(j>0) **DO** [LA.elem[k]:=LB.elem[j];k:=k-1;j:=j-1;]

LA.last:=m+n;

**ENDP**;

[算法讨论]算法中数据移动是主要操作。在最佳情况下（LB的最小元素大于LA的最大元素），仅将LB的n个元素移（拷贝）到LA中，时间复杂度为O（n），最差情况，LA的所有元素都要移动，时间复杂度为O（m+n）。因数据合并到LA中，所以在退出第一个**WHILE**循环后，只需要一个**WHILE**循环，处理LB中剩余元素。第二个循环只有在LB有剩余元素时才执行，而在LA有剩余元素时不执行。本算法利用了题目中“线性表空间足够大”的条件，“最大限度的避免移动元素”，是“一种高效算法”。

5．[题目分析]本题实质上是一个排序问题，要求“不得使用除该链表结点以外的任何链结点空间”。链表上的排序采用直接插入排序比较方便，即首先假定第一个结点有序，然后，从第二个结点开始，依次插入到前面有序链表中，最终达到整个链表有序。

LinkedList LinkListSort(LinkedList list)

∥list是不带头结点的线性链表，链表结点构造为data和link两个域，data是数据域，link是指针域。本算法将该链表按结点数据域的值的大小，从小到大重新链接。

{p=list->link; ∥p是工作指针，指向待排序的当前元素。

list->link=null;∥假定第一个元素有序，即链表中现只有一个结点。

**while**(p!=null)

{r=p->link; ∥r是p的后继。

q=list;

**if**(q->data>p->data)∥处理待排序结点p比第一个元素结点小的情况。

{p->link=list;

list=p;∥链表指针指向最小元素。

}

**else**∥查找元素值最小的结点。

{**while**(q->link!=null&&q->link->data<p->data)q=q->link;

p->link=q->link;∥将当前排序结点链入有序链表中。

q->link=p; }

p=r;∥p指向下个待排序结点。

}

}

[算法讨论]算法时间复杂度的分析与用顺序存储结构时的情况相同。但顺序存储结构将第i(i>1)个元素插入到前面第1至第i-1个元素的有序表时，是将第i个元素先与第i-1个元素比较。而在链表最佳情况均是和第一元素比较。两种存储结构下最佳和最差情况的比较次数相同，在链表情况下，不移动元素，而是修改结点指针。

另一说明是，本题中线性链表list不带头结点，而且要求“不得使用除该链表以外的任何链结点空间“，所以处理复杂，需要考虑当前结点元素值比有序链表第一结点的元素值还小的情况，这时要修改链表指针list。如果list是头结点的指针，则相应处理要简单些，其算法片段如下：

p=list->link;∥p指向第一元素结点。

list->link=null;∥有序链表初始化为空

**while**(p!=null)

{r=p->link;∥保存后继

q=list;

**while**(q->link!=null && q->link->data<p->data)q=q->link;

p->link=q->link;

q->link=p;

q=r;

}

6．[题目分析]本题明确指出单链表带头结点，其结点数据是正整数且不相同，要求利用直接插入原则把链表整理成递增有序链表。这就要求从第二结点开释，将各结点依次插入到有序链表中。

LinkedList LinkListInsertSort(LinkedList la)

∥la是带头结点的单链表，其数据域是正整数。本算法利用直接插入原则将链表整理成递增的有序链表。

{**if**(la->next!=null)∥链表不为空表。

{p=la->next->next;∥p指向第一结点的后继。

la->next->next=null;∥直接插入原则认为第一元素有序，然后从第二元素起依次插入。

**while**(p!=null)

{r=p->next;∥暂存p的后继。

q=la;

**while**(q->next!=null&&q->next->data<p->data)q=q->next;∥查找插入位置。

p->next=q->next;∥将p结点链入链表。

q->next=p;

p=r;

}

与本题有类似叙述的题的解答：

（1）本题也是链表排序问题，虽没象上题那样明确要求“利用直接插入的原则”来排序，仍可用上述算法求解，这里不再赘述。

7．[题目分析]本题要求将一个链表分解成两个链表，两个链表都要有序，两链表建立过程中不得使用NEW过程申请空间，这就是要利用原链表空间，随着原链表的分解，新建链表随之排序。

**PROC** discreat(VAR listhead,P,Q:linkedlist)

∥listhead是单链表的头指针，链表中每个结点由一个整数域DATA和指针域NEXT组成。本算法将链表listhead分解成奇数链表和偶数链表，分解由P和Q指向，且P和Q链表是有序的。

P:=NIL;Q:=NIL;∥P和Q链表初始化为空表。

s:=listhead;

**WHILE**(s<>NIL)**DO**

[r:=s^.NEXT; ∥暂存s的后继。

**IF** s^.DATA DIV 2=0 ∥处理偶数。

**THEN** **IF** P=NIL **THEN**[P:=s;P^.NEXT:=NIL;] ∥第一个偶数链结点。

**ELSE**[pre:=P;

**IF** pre^.DATA>s^.DATA **THEN**[s^.NEXT:=pre;P:=s;∥插入当前最小值结点修改头指针]

**ELSE**[**WHILE** pre^.NEXT<>NIL **DO**

**IF** pre^.NEXT^.DATA<s^.DATA **THEN** pre:=pre^.NEXT;∥查找插入位置。

s^.NEXT:=pre^.NEXT; ∥链入此结点。

pre^.NEXT:=s;

]

]

**ELSE**∥处理奇数链。

**IF** Q=NIL **THEN**[Q:=s;Q^.NEXT:=NIL;] ∥第一奇数链结点。

**ELSE**[pre:=Q;

**IF** pre^.DATA>s^.DATA **THEN**[s^.NEXT:=pre; Q:=s; ]∥修改头指针。

**ELSE**[**WHILE** pre^.NEXT<>NIL **DO** ∥查找插入位置。

**IF** pre^.NEXT^.DATA<s^.DATA **THEN** pre:=pre^.NEXT;

s^.NEXT:=pre^.NEXT;∥链入此结点。

pre^.NEXT:=s;

]

]∥结束奇数链结点

s:=r;∥s指向新的待排序结点。

]∥结束“**WHILE**(s<>NIL)**DO**”

**ENDP**；∥结束整个算法。

[算法讨论]由于算法要求“不得使用NEW过程申请空间，也没明确指出链表具有头结点，所以上述算法复杂些，它可能需要在第一个结点前插入新结点，即链表的头指针会发生变化。如有头结点，算法不必单独处理在第一个结点前插入结点情况，算法会规范统一，下面的（1）是处理带头结点的例子。算法中偶数链上结点是靠数据整除2等于0（DATA DIV 2=0）判断的。

类似本题的其它题解答如下：

（1）[题目分析]本题基本类似于上面第7题，不同之处有二。一是带头结点，二是分解后的两个链表，一个是数据值小于0，另一个是数据值大于0。由于没明确要求用类PASCAL书写算法，故用C书写如下。

**void** DisCreat1(LinkedList A)

∥A是带头结点的单链表，链表中结点的数据类型为整型。本算法将A分解成两个单链表B和C，B中结点的数据小于零，C中结点的数据大于零。

{B=A;

C=(LinkedList )malloc(**sizeof**(LNode));∥为C申请结点空间。

C->next=null ∥C初始化为空表。

p=A->next; ∥p为工作指针。

B->next=null; ∥B表初始化。

**while**(p!=null)

{r=p->next; ∥暂存p的后继。

**if** (p->data<0)∥小于0的放入B表。

{p->next=B->next; B->next=p; }∥将小于0的结点链入B表。

**else** {p->next=C->next; C->next=p; }

p=r;∥p指向新的待处理结点。

}

}∥算法结束。

[算法讨论]因为本题并未要求链表中结点的数据值有序，所以算法中采取最简单方式：将新结点前插到头结点后面（即第一元素之前）。

1. 本题同上面第7题，除个别叙述不同外，本质上完全相同，故不再另作解答。

（3）[题目分析]本题中的链表有头结点，分解成表A和表B，均带头结点。分解后的A表含有原表中序号为奇数的元素，B表含有原A表中序号为偶数的元素。由于要求分解后两表中元素结点的相对顺序不变，故采用在链表尾插入比较方便，这使用一指向表尾的指针即可方便实现。

**void** DisCreat3(LinkedList A)

∥A是带头结点的单链表，本算法将其分解成两个带头结点的单链表，A表中含原表中序号为奇数

∥的结点，B表中含原表中序号为偶数的结点。链表中结点的相对顺序同原链表。

{i=0;∥i记链表中结点的序号。

B=(LinkedList)malloc(**sizeof**(LNode);∥创建B表表头。

B->next=null; ∥B表的初始化。

LinkedList ra,rb;∥ra和rb将分别指向将创建的A表和B表的尾结点。

ra=A;rb=B;

p=A->next; ∥p为链表工作指针，指向待分解的结点。

A->next=null; ∥置空新的A表

**while**(p!=null)

{r=p->next; ∥暂存p的后继。

i++;

**if**(i%2==0) ∥处理原序号为偶数的链表结点。

{p->next=rb->next;∥在B表尾插入新结点；

rb->next=p; rb=p;∥rb指向新的尾结点；

}

**else**∥处理原序号为奇数的结点。

{p->next=ra->next; ra->next=p; ra=p; }

p=r; ∥将p恢复为指向新的待处理结点。

}∥算法结束

8．[题目分析]题目要求重排n个元素且以顺序存储结构存储的线性表，使得所有值为负数的元素移到正数元素的前面。这可采用快速排序的思想来实现，只是提出暂存的第一个元素（枢轴）并不作为以后的比较标准，比较的标准是元素是否为负数。

**int** Rearrange（SeqList a; **int** n)

∥a是具有n个元素的线性表，以顺序存储结构存储，线性表的元素是整数。本算法重排线性表a，

∥使所有值为负数的元素移到所有值为正数的数的前面。

{i=0; j=n-1; ∥ i,j为工作指针（下标），初始指向线性表a的第1个和第n个元素。

t=a[0]; ∥暂存枢轴元素。

**while**(i<j)

{**while**(i<j && a[j]>=0) j--; ∥ 若当前元素为大于等于零，则指针前移。

**if**(i<j){a[i]=a[j];i++;} ∥ 将负数前移。

**while**(i<j &&a[i]<0)i++; ∥ 当前元素为负数时指针后移。

**if**(i<j) a[j--]=a[i]; ∥ 正数后移。

}

a[i]=t; ∥将原第一元素放到最终位置。

}

[算法讨论] 本算法时间复杂度为O（n）。算法只是按题目要求把正负数分开，如要求统计负数和大于等于零的个数，则最后以t来定。如t为负数，则0至i共i+1个负数，n-1-i个正数（包括零）。另外，题目并未提及零的问题，笔者将零放到正数一边。对此问题的扩充是若元素包含正数、负数和零，并要求按负数、零、正数的顺序重排线性表，统计负数、零、正数的个数。请读者利用上面解题思想自行解答。

类似本题的选了5 个题，其解答如下：

（1）与上面第8题不同的是，这里要求以an为参考元素，将线性表分成左右两部分。左半部分的元素都小于等于an，右半部分的元素都大于an，an位于分界位置上。其算法主要片段语句如下:

i=1;j=n;

t=a[n]; ∥暂存参考元素。

**while**(i<j)

{**while**(i<j && a[i]<=t) i++; ∥当前元素不大于参考元素时，指针i后移。

**if**(i<j) a[j--]=a[i]; ∥将大于参考元素的元素后移。

**while**(i<j && a[j]>t) j--; ∥当前元素大于参考元素时指针前移。

**if**(i<j) a[i++]=a[j]; ∥将小于参考元素的当前元素前移。

}

a[i]=t; ∥参考元素置于分界位置。

(2) [题目分析]本题要求将线性表A分成B和C两个表，表B和表C不另占空间，而是利用表A的空间，其算法与第8题相同。这里仅把表B和表C另设空间的算法解答如下：

**void** Rearrange2(**int** A[],B[],C[])

∥线性表A有n个整型元素，顺序存储。本算法将A拆成B和C 两个表，B中存放大于

∥等于零的元素，C中存放小于零的元素。

{i=0; ∥i，j，k是工作指针，分别指向A、B和C表的当前元素。

j=k=-1; ∥j，k初始化为-1。

**while**(i<n)

{**if**(A[i]<0) C[++k]=A[i++]; ∥将小于零的元素放入C表。

**else** B[++j]=A[i++]; ∥将大于零的元素放入B表。

[算法讨论]本题用一维数组存储线性表，结果线性表B和C中分别有j+1和k+1个元素。若采用教材中的线性表，则元素的表示作相应改变，例如A.elem[i]，而最后B和C表应置上表的长度，如B.length=j和C.length=k。

(3) 本题与第8题本质上相同，第8题要求分开正数和负数，这里要求分开奇数和偶数，判别方式是a[i]%2==0，满足时为偶数，反之为奇数。

(4) 本题与第8题相同，只是叙述不同。

(5) 本题与第8题基本相同，不同之处在于这里的分界元素是整数19（链表中并不要求一定有19）。本题要求用标准pascal描述算法，如下所示。

**TYPE** arr=**ARRAY**[1..1000] **OF**  integer；

**VAR**  a：arr；

**PROCEDURE** Rearrange5（VAR a：arr）；

∥a是n（设n=1000）个整数组成的线性表，用一维数组存储。本算法将n个元素中所有大于等于19的整数放在所有小于19的整数之后。

**VAR** i,j,t：integer；

**BEGIN**

i:=1；j:=n；t:=a[1] ；∥i,j指示顺序表的首尾元素的下标，t暂存分界元素

**WHILE**（i<j）**DO**

**BEGIN**

**WHILE** （i<j）**AND**（a[j]>=19） **DO** j:=j-1；

**IF**（i<j）**THEN** **BEGIN** A[i]:=A[j]；i:=i+1 **END**；

**WHILE** （i<j）**AND**（a[i] <19） **DO** i:=i+1；

**IF**（i<j）**THEN** **BEGIN** A[j]:=A[i]；j:=j-1 **END**；

**END**；

a[i]:=t；

**END;**

[算法讨论] 分界元素t放入a[i]，而不论它的值如何。算法中只用了一个t中间变量，符合空间复杂度O(1)的要求。算法也满足时间复杂度O(n)的要求。

9．[题目分析] 本题要求在单链表中删除最小值结点。单链表中删除结点，为使结点删除后不出现“断链”，应知道被删结点的前驱。而“最小值结点”是在遍历整个链表后才能知道。所以算法应首先遍历链表，求得最小值结点及其前驱。遍历结束后再执行删除操作。

LinkedList Delete（LinkedList L）

∥L是带头结点的单链表，本算法删除其最小值结点。

{p=L->next； ∥p为工作指针。指向待处理的结点。假定链表非空。

pre=L； ∥pre指向最小值结点的前驱。

q=p； ∥q指向最小值结点，初始假定第一元素结点是最小值结点。

**while**（p->next!=null）

{**if**（p->next->data<q->data）{pre=p；q=p->next；} ∥查最小值结点

p=p->next； ∥指针后移。

}

pre->next=q->next；∥从链表上删除最小值结点

free（q）； ∥释放最小值结点空间

}∥结束算法delete。

[算法讨论] 算法中函数头是按本教材类C描述语言书写的。原题中**void** delete（linklist &L），是按C++的“引用”来写的，目的是实现变量的“传址”，克服了C语言函数传递只是“值传递”的缺点。

10．[题目分析] 本题要求将链表中数据域值最小的结点移到链表的最前面。首先要查找最小值结点。将其移到链表最前面，实质上是将该结点从链表上摘下（不是删除并回收空间），再插入到链表的最前面。

LinkedList delinsert（LinkedList list）

∥list是非空线性链表，链结点结构是（data，link），data是数据域，link是链域。

∥本算法将链表中数据域值最小的那个结点移到链表的最前面。

{p=list->link；∥p是链表的工作指针

pre=list； ∥pre指向链表中数据域最小值结点的前驱。

q=p； ∥q指向数据域最小值结点，初始假定是第一结点

**while** （p->link!=null）

{**if**（p->link->data<q->data）{pre=p；q=p->link；} ∥找到新的最小值结点；

p=p->link；

}

**if** (q!=list->link) ∥若最小值是第一元素结点，则不需再操作

{pre->link=q->link； ∥将最小值结点从链表上摘下；

q->link= list->link；∥将q结点插到链表最前面。

list->link=q；

}

}∥算法结束

[算法讨论] 算法中假定list带有头结点，否则，插入操作变为q->link=list；list=q。

11．[题目分析] 知道双向循环链表中的一个结点，与前驱交换涉及到四个结点（p结点，前驱结点，前驱的前驱结点，后继结点）六条链。

**void** Exchange（LinkedList p）

∥p是双向循环链表中的一个结点，本算法将p所指结点与其前驱结点交换。

{q=p->llink；

q->llink->rlink=p； ∥p的前驱的前驱之后继为p

p->llink=q->llink； ∥p的前驱指向其前驱的前驱。

q->rlink=p->rlink； ∥p的前驱的后继为p的后继。

q->llink=p； ∥p与其前驱交换

p->rlink->llink=q； ∥p的后继的前驱指向原p的前驱

p->rlink=q； ∥p的后继指向其原来的前驱

}∥算法exchange结束。

12．[题目分析] 顺序存储的线性表递增有序，可以顺序查找，也可折半查找。题目要求“用最少的时间在表中查找数值为x的元素”，这里应使用折半查找方法。

**void** SearchExchangeInsert（ElemType a[]；ElemType x）

∥a是具有n个元素的递增有序线性表，顺序存储。本算法在表中查找数值为x的元素，如查到则与其后继交换位置；如查不到，则插入表中，且使表仍递增有序。

{ low=0；high=n-1； ∥low和high指向线性表下界和上界的下标

**while**（low<=high）

{mid=（low+high）/2； ∥找中间位置

**if**（a[mid]==x） **break**； ∥找到x，退出**while**循环。

**else** **if** （a[mid] <x） low=mid+1；∥到中点mid的右半去查。

**else** high=mid-1； ∥到中点mid的左部去查。

}

**if**（a[mid]==x && mid!=n）∥ 若最后一个元素与x相等，则不存在与其后继交换的操作。

{t=a[mid]；a[mid]=a[mid+1]；a[mid+1]=t；} ∥ 数值x与其后继元素位置交换。

**if**（low>high） ∥查找失败，插入数据元素x

{**for**（i=n-1；i>high；i--） a[i+1]=a[i]；∥后移元素。

a[i+1]=x；∥插入x。

} ∥结束插入

}∥结束本算法。

[算法讨论] 首先是线性表的描述。算法中使用一维数组a表示线性表，未使用包含数据元素的一维数组和指示线性表长度的结构体。若使用结构体，对元素的引用应使用a.elem[i]。另外元素类型就假定是ElemType，未指明具体类型。其次，C中一维数组下标从0开始，若说有n个元素的一维数组，其最后一个元素的下标应是n-1。第三，本算法可以写成三个函数，查找函数，交换后继函数与插入函数。写成三个函数显得逻辑清晰，易读。

13．[题目分析] 判断链表中数据是否中心对称，通常使用栈。将链表的前一半元素依次进栈。在处理链表的后一半元素时，当访问到链表的一个元素后，就从栈中弹出一个元素，两元素比较，若相等，则将链表中下一元素与栈中再弹出元素比较，直至链表到尾。这时若栈是空栈，则得出链表中心对称的结论；否则，当链表中一元素与栈中弹出元素不等时，结论为链表非中心对称，结束算法的执行。

**int**  dc（LinkedList h，**int** n）

∥ h是带头结点的n个元素单链表，链表中结点的数据域是字符。本算法判断链表是否是中心对称。

{**char** s[]; **int** i=1；∥i记结点个数， s字符栈

p=h->next；∥p是链表的工作指针，指向待处理的当前元素。

**for**（i=1；i<=n/2；i++）∥ 链表前一半元素进栈。

{s[i]=p->data；p=p->next；}

i--; ∥恢复最后的i值

**if**（n%2==1）p=p->next；} ∥若n是奇数，后移过中心结点。

**while**（p!=null && s[i]==p->data）{i--；p=p->next；} ∥测试是否中心对称。

**if**（p==null）**return**（1）；∥链表中心对称

**else** **return**（0）； ∥链表不中心对称

}∥算法结束。

[算法讨论] 算法中先将“链表的前一半”元素（字符）进栈。当n为偶数时，前一半和后一半的个数相同；当n为奇数时，链表中心结点字符不必比较，移动链表指针到下一字符开始比较。比较过程中遇到不相等时，立即退出**while**循环，不再进行比较。

14．[题目分析] 在单链表中删除自第i个元素起的共len个元素，应从第1个元素起开始计数，记到第i个时开始数len个，然后将第i-1个元素的后继指针指向第i+len个结点，实现了在A链表中删除自第i个起的len个结点。这时应继续查到A的尾结点，得到删除元素后的A链表。再查B链表的第j个元素，将A链表插入之。插入和删除中应注意前驱后继关系，不能使链表“断链”。另外，算法中应判断i，len和j的合法性。

LinkedList DelInsert（LinkedList heada**，**headb，**int** i，j，len）

∥heada和headb均是带头结点的单链表。本算法删除heada链表中自第i个元素起的共len个元素，然后将单链表heada插入到headb的第j个元素之前。

{**if**（i<1 || len<1 || j<1）{printf（“参数错误\n”）；exit（0）；}∥参数错，退出算法。

p=heada；∥p为链表A的工作指针，初始化为A的头指针，查到第i个元素时，p指向第i-1个元素

k=0；∥计数

**while**（p!=null && k<i-1）∥查找第i个结点。

{k++；p=p->next；}

**if**（p==null）{printf（“给的%d太大\n”,i）；exit（0）；} ∥i太大，退出算法

q=p->next；∥q为工作指针，初始指向A链表第一个被删结点。

k=0；

**while**（q!=null && k<len）{k++；u=q，q=q->next；free（u）；} ∥删除结点，后移指针。

**if**（k<len）{printf（“给的%d太大\n”,len）；exit（0）；}

p->next=q；∥A链表删除了len个元素。

**if** (heada->next!=null) ∥heada->next=null说明链表中结点均已删除，无需往B表插入

{**while**（p->next!=null）p= p->next；∥找A的尾结点。

q=headb；∥q为链表B的工作指针。

k=0；∥计数

**while**（q!=null && k<j-1） ∥查找第j个结点。

{k++；q= q->next；} ∥查找成功时，q指向第j-1个结点

**if**（q==null）{printf（“给的%d太大\n”,j）；exit（0）；}

p->next=q->next； ∥将A链表链入

q->next=heada->next； ∥A的第一元素结点链在B的第j-1个结点之后

}//**if**

free（heada）；∥释放A表头结点。

}∥算法结束。

与本题类似的题的解答如下：

（1）本题与第14题基本相同，不同之处仅在于插入B链表第j个元素之前的，不是删除了len个元素的A链表，而是被删除的len个元素。按照上题，这len个元素结点中第一个结点的指针p->next，查找从第i个结点开始的第len个结点的算法修改为：

k=1；q=p->next； //q指向第一个被删除结点

**while**（q!=null && k<len） ∥查找成功时，q指向自i起的第len个结点。

{k++；q= q->next；}

**if**（k<len） {printf（“给的%d太大\n”,len）；exit（0）；}

15．[题目分析] 在递增有序的顺序表中插入一个元素x，首先应查找待插入元素的位置。因顺序表元素递增有序，采用折半查找法比顺序查找效率要高。查到插入位置后，从此位置直到线性表尾依次向后移动一个元素位置，之后将元素x插入即可。

**void** Insert（ElemType A[]，**int** size, ElemType x）

∥ A是有size个元素空间目前仅有num（num<size）个元素的线性表。本算法将元素x插入到线性表中，并保持线性表的有序性。

{low=1；high=num； //题目要求下标从1开始

**while**（low<=high）∥对分查找元素x的插入位置。

{mid=（low+high）/2；

**if**（A[mid]==x）{low=mid+1；**break**；}

**else** **if**（A[mid]>x）high=mid-1 ；**else** low=mid+1 ；

}

**for**（i=num；i>=low；i--） A[i+1]=A[i]；∥元素后移。

A[i+1]=x； ∥将元素x插入。

}算法结束。

[算法讨论] 算法中当查找失败（即线性表中无元素x）时，变量low在变量high的右面（low=high+1）。移动元素从low开始，直到num为止。特别注意不能写成**for**（i=low；i<=num；i++）A[i+1]=A[i]，这是一些学生容易犯的错误。另外，题中未说明若表中已有值为x的元素时不再插入，故安排在A[mid]= =x时，用low（=mid+1）记住位置，以便后面统一处理。查找算法时间复杂度为O（logn），而插入时的移动操作时间复杂度为O（n），若用顺序查找，则查找的时间复杂度亦为O（n）。

类似本题的其它题的解答：

（1）[题目分析] 本题与上面15题类似，不同之处是给出具体元素值，且让编写turbo pascal程序，程序如下：

**PROGRAM** example（input，output）；

**TYPE** pointer=^node；

node=**RECORD**

data：integer；

next：pointer；

**END**；

**VAR** head，q：pointer；

**PROCEDURE** create（**VAR** la：pointer）；

**VAR** x：integer；

p,q：pointer；

**BEGIN**

new（la）；la^.next:=NIL；{建立头结点。}

read（x）；q:=la；{q用以指向表尾。}

**WHILE** **NOT** EOF **DO** {建立链表}

**BEGIN**

new（p）；p^.data:=x；p^.next:=q^.next；q^.next:=p；q:=p; read(x);

**END**；

**END**；

**PROCEDURE** insert（**VAR** la：pointer；s：pointer）；

**VAR** p,q：pointer；found：boolean；

**BEGIN**

p:= la^.next；{p为工作指针。}

q:=la；{q为p的前驱指针。}

found:=false；

**WHILE**（p<>NIL）**AND** **NOT** found

**IF**（p^.data<x）**THEN** **BEGIN** q:=p；p:= p^.next； **END**

**ELSE** found:=true；

s^.next:=p；{将s结点插入链表}

q^.next:=s；

**END**；

**BEGIN** {main}

writeln（“请按顺序输入数据，建立链表”）

create（head）；

writeln（“请输入插入数据”）

new（q）；

readln（q^.data）；

insert（head，q）；

**END**.{程序结束}

[程序讨论] 在建立链表时，输入数据依次为12，13，21，24，28，30，42，键入CTRL-Z，输入结束。“插入数据”输26即可。本题编写的是完整的pascal程序。

16．[题目分析] 将具有两个链域的单循环链表，改造成双向循环链表，关键是控制给每个结点均置上指向前驱的指针，而且每个结点的前驱指针置且仅置一次。

**void** StoDouble（LinkedList la）

∥la是结点含有pre，data，link三个域的单循环链表。其中data为数据域,pre为空指针域，link是指向后继的指针域。本算法将其改造成双向循环链表。

{**while**（la->link->pre==null）

{la->link->pre=la； ∥将结点la后继的pre指针指向la。

la=la->link； ∥la指针后移。

}

} ∥算法结束。

[算法讨论] 算法中没有设置变量记住单循环链表的起始结点，至少省去了一个指针变量。当算法结束时，la恢复到指向刚开始操作的结点，这是本算法的优点所在。

17．[题目分析] 求两个集合A和B的差集A-B，即在A中删除A和B中共有的元素。由于集合用单链表存储，问题变成删除链表中的结点问题。因此，要记住被删除结点的前驱，以便顺利删除被删结点。两链表均从第一元素结点开始，直到其中一个链表到尾为止。

**void** Difference（LinkedList A，B，\*n）

∥A和B均是带头结点的递增有序的单链表，分别存储了一个集合，本算法求两集合的差集，存储于单链表A中，\*n是结果集合中元素个数，调用时为0

{p=A->next； ∥p和q分别是链表A和B的工作指针。

q=B->next； pre=A； ∥pre为A中p所指结点的前驱结点的指针。

**while**（p!=null && q!=null）

**if**（p->data<q->data）{pre=p；p=p->next；\*n++；} ∥ A链表中当前结点指针后移。

**else** **if**（p->data>q->data）q=q->next； ∥B链表中当前结点指针后移。

**else** {pre->next=p->next； ∥处理A，B中元素值相同的结点，应删除。

u=p； p=p->next； free（u）；} ∥删除结点

18．[题目分析] 本题要求对单链表结点的元素值进行运算，判断元素值是否等于其序号的平方减去其前驱的值。这里主要技术问题是结点的序号和前驱及后继指针的正确指向。

**int**  Judge（LinkedList la）

∥la是结点的元素为整数的单链表。本算法判断从第二结点开始，每个元素值是否等于其序号的平方减去其前驱的值，如是返回true；否则，返回false。

{p=la->next->next；∥p是工作指针，初始指向链表的第二项。

pre=la->next； ∥pre是p所指结点的前驱指针。

i=2； ∥i是la链表中结点的序号，初始值为2。

**while**（p!=null）

**if**（p->data==i\*i-pre->data）{i++；pre=p；p=p->next；} ∥结点值间的关系符合题目要求

**else** **break**； ∥当前结点的值不等于其序号的平方减去前驱的值。

**if**（p!=null）**return**（false）； ∥未查到表尾就结束了。

**else** **return**（true）； ∥成功返回。

}∥算法结束。

[算法讨论]本题不设头结点也无影响。另外，算法中还可节省前驱指针pre，其算法片段如下：

p=la；∥假设无头结点，初始p指向第一元素结点。

i=2；

**while**（p->next!=null） ∥初始p->next指向第二项。

**if**（p->next->data= =i\*i-p->data）

{i++；p=p->next；}

**if**（p->next!=null）**return**（false）；∥失败

**else** **return**（true）； ∥成功

19．[题目分析] 本题实质上是一个模式匹配问题，这里匹配的元素是整数而不是字符。因两整数序列已存入两个链表中，操作从两链表的第一个结点开始，若对应数据相等，则后移指针；若对应数据不等，则A链表从上次开始比较结点的后继开始，B链表仍从第一结点开始比较，直到B链表到尾表示匹配成功。A链表到尾B链表未到尾表示失败。操作中应记住A链表每次的开始结点，以便下趟匹配时好从其后继开始。

**int**  Pattern（LinkedList A，B）

∥A和B分别是数据域为整数的单链表，本算法判断B是否是A的子序列。如是，返回1；否则，返回0表示失败。

{p=A； ∥p为A链表的工作指针，本题假定A和B均无头结点。

pre=p； ∥pre记住每趟比较中A链表的开始结点。

q=B； ∥q是B链表的工作指针。

**while**（p && q）

**if**（p->data==q->data） {p=p->next；q=q->next；}

**else**{pre=pre->next；p=pre； ∥A链表新的开始比较结点。

q=B；} ∥q从B链表第一结点开始。

**if**（q==null）**return**（1）； ∥B是A的子序列。

**else** **return**（0）； ∥B不是A的子序列。

}∥算法结束。

20．[题目分析] 本题也是模式匹配问题，应先找出链表L2在链表L1中的出现，然后将L1中的L2倒置过来。设L2在L1中出现时第一个字母结点的前驱的指针为p，最后一个字母结点在L1中为q所指结点的前驱，则在保存p后继结点指针(s)的情况下，执行p->next=q。之后将s到q结点的前驱依次插入到p结点之后，实现了L2在L1中的倒置。

LinkedList PatternInvert（LinkedList L1，L2）

∥L1和L2均是带头结点的单链表，数据结点的数据域均为一个字符。本算法将L1中与L2中数据域相同的连续结点的顺序完全倒置过来。

{p=L1； ∥p是每趟匹配时L1中的起始结点前驱的指针。

q=L1->next； ∥q是L1中的工作指针。

s=L2->next； ∥s是L2中的工作指针。

**while**（p!=null && s!=null）

**if**（q->data==s->data）{q=q->next;s=s->next;} ∥对应字母相等，指针后移。

**else** {p=p->next；q=p->next；s=L2->next；} ∥失配时，L1起始结点后移，L2从首结点开始。

**if**（s==null）∥匹配成功，这时p为L1中与L2中首字母结点相同数据域结点的前驱，q为L1中与L2最后一个结点相同数据域结点的后继。

{r=p->next； ∥r为L1的工作指针，初始指向匹配的首字母结点。

p->next=q； ∥将p与q结点的链接。

**while**（r!=q）； ∥逐结点倒置。

{s=r->next； ∥暂存r的后继。

r->next=p->next；∥将r所指结点倒置。

p->next=r；

r=s； ∥恢复r为当前结点。

}

}

**else** printf（“L2并未在L1中出现”）；

} ∥算法结束。

[算法讨论] 本算法只讨论了L2在L1至多出现一次（可能没出现），没考虑在L1中多次出现的情况。若考虑多次出现，可在上面算法找到第一次出现后的q结点作L1中下次比较的第一字母结点，读者可自行完善之。

类似本题的另外叙述题的解答：

（1）[题目分析] 本题应先查找第i个结点，记下第i个结点的指针。然后从第i+1个结点起，直至第m（1<i<m）个结点止，依次插入到第i-1个结点之后。最后将暂存的第i个结点的指针指向第m结点，形成新的循环链表，结束了倒置算法。

LinkedList PatternInvert1（LinkedList L，**int**  i，m）

∥L是有m个结点的链表的头结点的指针。表中从第i（1<i<m）个结点到第m个结点构成循环部分链表，本算法将这部分循环链表倒置。

{**if**（i<1|| i>=m || m<4）{printf（“%d,%d参数错误\n”,i，m）；exit（0）；}

p=L->next->next； ∥p是工作指针，初始指向第二结点（已假定i>1）。

pre=L->next； ∥pre是前驱结点指针，最终指向第i-1个结点。

j=1； ∥计数器

**while**（j<i-1） ∥查找第i个结点。

{j++；pre=p；p=p->next；}∥查找结束，p指向第i个结点。

q=p； ∥暂存第i个结点的指针。

p=p->next； ∥p指向第i+1个结点，准备逆置。

j+=2； ∥上面**while**循环结束时，j=i-1，现从第i+1结点开始逆置。

**while**（j<=m）

{r=p->next； ∥暂存p的后继结点。

p->next=pre->next；∥逆置p结点。

pre->next=p；

p=r； ∥p恢复为当前待逆置结点。

j++； ∥计数器增1。

}

q->next=pre->next；∥将原第i个结点的后继指针指向原第m个结点。

[算法讨论] 算法中未深入讨论i，m，j的合法性，因题目的条件是m>3且1<i<m。因此控制循环并未用指针判断（如一般情况下的p!=null），结束循环也未用指针判断。注意最后一句q->next=pre->next，实现了从原第i个结点到原第m个结点的循环。最后pre->next正是指向原第m个结点，不可用p->next代替pre->next。

21．[题目分析] 顺序存储结构的线性表的逆置，只需一个变量辅助空间。算法核心是选择循环控制变量的初值和终值。

**void** SeqInvert（ElemType a[ ]，**int** n）

∥a是具有n个元素用一维数组存储的线性表，本算法将其逆置。

{**for**（i=0；i<=（n-1）/2；i++）

{t=a[i]；a[i]= a[n-1-i]；a[n-1-i]=t；}

}∥算法结束

[算法讨论] 算法中循环控制变量的初值和终值是关键。C中数组从下标0开始，第n个元素的下标是n-1。因为首尾对称交换，所以控制变量的终值是线性表长度的一半。当n为偶数，“一半”恰好是线性表长度的二分之一；若n是奇数，“一半”是小于n/2的最大整数，这时取大于1/2的最小整数的位置上的元素，恰是线性表中间位置的元素，不需要逆置。另外，由于pascal数组通常从下标1开始，所以，上下界处理上略有不同。这点请读者注意。

类似本题的其它题的解答：

这一组又选了6个题，都是单链表（包括单循环链表）的逆置。链表逆置的通常作法是：将工作指针指向第一个元素结点，将头结点的指针域置空。然后将链表各结点从第一结点开始直至最后一个结点，依次前插至头结点后，使最后插入的结点成为链表的第一结点，第一个插入的结点成为链表的最后结点。

（1）要求编程实现带头结点的单链表的逆置。首先建立一单链表，然后逆置。

**typedef** **struct** node

{**int**  data；∥假定结点数据域为整型。

**struct** node \*next；

}node,\*LinkedList；

LinkedList creat（ ）

{LinkedList head，p

**int**  x；

head=（LinkedList）malloc（**sizeof**（node））；

head->next=null； /\*设置头结点\*/

scanf（“%d”，&x）；

**while**（x!=9999） /\*约定输入9999时退出本函数\*/

{p=（LinkedList）malloc（**sizeof**（node））；

p->data=x；

p->next=head->next；/\* 将新结点链入链表\*/

head->next=p；

scanf（“%d”，&x）；

}

**return**（head）；

}∥结束creat函数。

LinkedList invert1（LinkedList head）

/\*逆置单链表\*/

{LinkedList p=head->next; /\*p为工作指针\*/

head->next=null；

**while**（p!=null）

{r=p->next； /\*暂存p的后继\*/

p->next=head->next；

head->next=p；

p=r；

}

**return**（head）；

}/\*结束invert1函数\*/

main（）

{LinkedList la；

la=creat( ）； /\*生成单链表\*/

la=invert1（la）；/\*逆置单链表\*/

}

（2）本题要求将数据项递减有序的单链表重新排序，使数据项递增有序，要求算法复杂度为O（n）。虽没说要求将链表逆置，这只是叙述不同，本质上是将单链表逆置，现编写如下：

LinkedList invert2（LinkedList la）

∥la是带头结点且数据项递减有序的单链表，本算法将其排列成数据项递增有序的单链表。

{p=la->next； /\*p为工作指针\*/

la->next=null；

**while**（p!=null）

{r=p->next； /\*暂存p的后继。\*/

p->next=la->next；/\*将p结点前插入头结点后。\*/

la->next=p；p=r；

}

}∥结束算法

（3）本题要求倒排循环链表，与上面倒排单链表处理不同之处有二：一是初始化成循环链表而不是空链表；二是判断链表尾不用空指针而用是否是链表头指针。算法中语句片段如下：

p=la->next； ∥p为工作指针。

la->next=la； ∥初始化成空循环链表。

**while**（p!=la） ∥当p=la时循环结束。

{r=p->next； ∥暂存p的后继结点

p->next=la->next；∥逆置

la->next=p； p=r；

}

（4）不带头结点的单链表逆置比较复杂，解决方法可以给加上头结点：

la=（LinkedList）malloc（**sizeof**（node））；

la->next=L；

之后进行如上面（2）那样的逆置，最后再删去头结点：

L=la->next； ∥L是不带头结点的链表的指针。

free（la）； ∥释放头结点。

若不增加头结点，可用如下语句片段：

p=L->next； ∥p为工作指针。

L->next=null；∥第一结点成为尾结点。

**while**（p!=null）

{r=p->next；

p->next=L；∥将p结点插到L结点前面。

L=p； ∥L指向新的链表“第一”元素结点。

p=r；

}

（5）同（4），只是叙述有异。

（6）同（2），差别仅在于叙述不同。

22．[题目分析] 在无序的单链表上，查找最小值结点，要查遍整个链表，初始假定第一结点是最小值结点。当找到最小值结点后，判断数据域的值是否是奇数，若是，则“与其后继结点的值相交换”即仅仅交换数据域的值，用三个赋值语句即可交换。若与后继结点交换位置，则需交换指针，这时应知道最小值结点的前驱。至于删除后继结点，则通过修改最小值结点的指针域即可。

[算法设计]

**void** MiniValue（LinkedList la）

∥la是数据域为正整数且无序的单链表，本算法查找最小值结点且打印。若最小值结点的数值是奇数，则与后继结点值交换；否则，就删除其直接后继结点。

{p=la->next； ∥设la是头结点的头指针，p为工作指针。

pre=p； ∥pre指向最小值结点，初始假定首元结点值最小。

**while**（p->next!=null） ∥p->next是待比较的当前结点。

{**if**（p->next->data<pre->data）pre=p->next；

p=p->next； ∥后移指针

}

printf（“最小值=%d\n”，pre->data）；

**if**（pre->data%2!=0） ∥处理奇数

**if**（pre->next!=null）∥若该结点没有后继，则不必交换

{t= pre->data；pre->data=pre->next->data；pre->next->data=t；}∥交换完毕

**else**∥处理偶数情况

**if**（pre->next!=null）∥若最小值结点是最后一个结点，则无后继

{u=pre->next；pre->next=u->next；free（u）；} ∥释放后继结点空间

23．[题目分析] 将一个结点数据域为字符的单链表，分解成含有字母字符、数字字符和其它字符的三个循环链表，首先要构造分别含有这三类字符的表头结点。然后从原链表第一个结点开始，根据结点数据域是字母字符、数字字符和其它字符而分别插入到三个链表之一的链表。注意不要因结点插入新建链表而使原链表断链。另外，题目并未要求链表有序，插入采用“前插法”，每次插入的结点均成为所插入链表的第一元素的结点即可。

**void** OneToThree（LinkedList L，la，ld，lo）

∥L是无头结点的单链表第一个结点的指针，链表中的数据域存放字符。本算法将链表L分解成含有英文字母字符、数字字符和其它字符的带头结点的三个循环链表。

{la=（LinkedList）malloc（**sizeof**（LNode））； ∥建立三个链表的头结点

ld=（LinkedList）malloc（**sizeof**（LNode））；

lo=（LinkedList）malloc（**sizeof**（LNode））；

la->next=la；ld->next=ld；lo->next=lo；∥置三个循环链表为空表

**while**（L!=null）∥分解原链表。

{r=L; L=L->next; ∥L指向待处理结点的后继

**if**（r->data>=‘a’&& r->data<=‘z’|| r->data>=‘A’&& r->data<=‘Z’）

{r->next=la->next； la->next=r；} ∥处理字母字符。

**else** **if**（r->data>=‘0’&& r->data<=‘9’）

{r->next=ld->next；ld->next=r；} ∥处理数字字符

**else** {r->next=lo->next；lo->next=r；} ∥处理其它符号。

}∥结束**while**（L!=null）。

}∥算法结束

[算法讨论] 算法中对L链表中每个结点只处理一次，时间复杂度O（n），只增加了必须的三个表头结点，符合题目“用最少的时间和最少的空间”的要求。

24．[题目分析] 在递增有序的线性表中，删除数值相同的元素，要知道被删除元素结点的前驱结点。

LinkedList DelSame（LinkedList la）

∥la是递增有序的单链表，本算法去掉数值相同的元素，使表中不再有重复的元素。

{pre=la->next；∥pre是p所指向的前驱结点的指针。

p=pre->next；∥p是工作指针。设链表中至少有一个结点。

**while**（p!=null）

**if**（p->data==pre->data） ∥处理相同元素值的结点

{u=p；p=p->next；free（u）；} ∥释放相同元素值的结点

**else** {pre->next=p；pre=p；p=p->next；} ∥处理前驱，后继元素值不同

pre->next=p；∥置链表尾。

}∥DelSame

[算法讨论] 算法中假设链表至少有一个结点，即初始时pre不为空，否则p->next无意义。算法中最后pre->next=p是必须的，因为可能链表最后有数据域值相同的结点，这些结点均被删除，指针后移使p=null而退出**while**循环，所以应有pre->next=p使链表有尾。若链表尾部没数据域相同的结点，pre和p为前驱和后继，pre->next=p也是对的。

顺便提及，题目应叙述为非递减有序，因为“递增”是说明各结点数据域不同，一个值比一个值大，不会存在相同值元素。

25．[题目分析] 建立递增有序的顺序表，对每个输入数据，应首先查找该数据在顺序表中的位置，若表中没有该元素则插入之，如已有该元素，则不再插入，为此采用折半查找方法。

**FUNC** BinSearch（**VAR** a：sqlisttp；x：integer）：integer；

∥在顺序表a中查找值为x的元素，如查找成功，返回0值，如x不在a中，则返回查找失败时的较大下标值。

low:=1；high:=a.last；found:=false；

**WHILE**（low<=high）**AND** **NOT** found **DO**

[mid:=（low+high）**DIV** 2；

**IF** a.elem[mid]=x **THEN** found:=true

**ELSE** **IF** a.elem[mid]>x **THEN** high:=mid-1 **ELSE** low:=mid+1；

]

**IF** found=true **THEN** **return**（0）

**ELSE** **return**（low）；∥当查找失败时，low=high+1。

**ENDF**；∥结束对分查找函数。

**PROC** create（**VAR** L：sqlisttp）

∥本过程生成顺序表L。

L.last:=0； ∥顺序表L初始化。

read（x）；

**WHILE** x<>9999 **DO** ∥设x=9999时退出输入

[k:=binsearch（L，x）；∥去查找x元素。

**IF** k<>0 ∥不同元素才插入

**THEN** [**FOR** i:=L.last **DO**WN**TO** k **DO** L.elem[i+1]:=L.elem[i]；

L.elem[k]=x；L.last:= L.last+1；∥插入元素x，线性表长度增1

]

read（x）；

]

**ENDP**；∥结束过程creat

26．[题目分析] 在由正整数序列组成的有序单链表中，数据递增有序，允许相等整数存在。确定比正整数x大的数有几个属于计数问题，相同数只计一次，要求记住前驱，前驱和后继值不同时移动前驱指针，进行计数。将比正整数x小的数按递减排序，属于单链表的逆置问题。比正整数x大的偶数从表中删除，属于单链表中结点的删除，必须记住其前驱，以使链表不断链。算法结束时，链表中结点的排列是：小于x的数按递减排列，接着是x（若有的话），最后是大于x的奇数。

**void** exam（LinkedList la，**int** x）

∥la是递增有序单链表，数据域为正整数。本算法确定比x大的数有几个；将比x小的数按递减排序，并将比x大的偶数从链表中删除。）

{p=la->next；q=p;∥p为工作指针 q指向最小值元素，其可能的后继将是>=x的第一个元素。

pre=la； ∥pre为p的前驱结点指针。

k=0； ∥计数（比x大的数）。

la->next=null；∥置空单链表表头结点。

**while**（p && p->data<x） ∥先解决比x小的数按递减次序排列

{r=p->next； ∥暂存后继

p->next=la->next；∥逆置

la->next=p；

p=r；∥恢复当前指针。退出循环时，r指向值>=x的结点。

}

q->next=p; pre=q; ∥pre指向结点的前驱结点

**while**（p->data==x）{pre=p; p=p->next;} ∥从小于x到大于x可能经过等于x

**while**（p） ∥以下结点的数据域的值均大于x

{k++; x=p->data; ∥下面仍用x表示数据域的值，计数

**if**（x % 2==0） ∥删偶数

{**while** (p->data==x)

{u=p；p=p->next；free(u)；}

pre->next=p; ∥拉上链

}

**else** ∥处理奇数

**while** (p->data==x)∥相同数只记一次

{pre->next=p；pre=p；p=p->next；}

}∥**while**(p)

printf（“比值%d大的数有%d个\n”，x，k）；

}∥算法exam结束

[算法讨论] 本题“要求用最少的时间和最小的空间”。本算法中“最少的时间”体现在链表指针不回溯，最小空间是利用了几个变量。在查比x大的数时，必须找到第一个比x大的数所在结点（因等于x的数可能有，也可能多个，也可能没有）。之后，计数据的第一次出现，同时删去偶数。

顺便指出，题目设有“按递增次序”的“有序单链表”，所给例子序列与题目的论述并不一致。

27．[题目分析] 单链表中查找任何结点，都必须从头指针开始。本题要求将指针p所指结点与其后继结点交换，这不仅要求知道p结点，还应知道p的前驱结点。这样才能在p与其后继结点交换后，由原p结点的前驱来指向原p结点的后继结点。

另外，若无特别说明，为了处理的方便统一，单链表均设头结点，链表的指针就是头结点的指针。并且由于链表指针具有标记链表的作用，也常用指针名冠以链表名称。如“链表head”既指的是链表的名字是head，也指出链表的头指针是head。

LinkedList Exchange（LinkedList HEAD，p）

∥HEAD是单链表头结点的指针，p是链表中的一个结点。本算法将p所指结点与其后继结点交换。

{q=head->next； ∥q是工作指针，指向链表中当前待处理结点。

pre=head； ∥pre是前驱结点指针，指向q的前驱。

**while**（q!=null && q!=p）{pre=q；q=q->next；} ∥未找到p结点，后移指针。

**if**（p->next==null）printf（“p无后继结点\n”）; ∥p是链表中最后一个结点，无后继。

**else**∥处理p和后继结点交换

{q=p->next； ∥暂存p的后继。

pre->next=q； ∥p前驱结点的后继指向p的后继。

p->next=q->next；∥p的后继指向原p后继的后继。

q->next=p ；∥原p后继的后继指针指向p。

}

}∥算法结束。

类似本题的其它题目的解答：

（1）与上面第27题基本相同，只是明确说明“p指向的不是链表最后那个结点。”

（2）与上面第27题基本相同，仅叙述不同，故不再作解答。

28．[题目分析] 本题链表结点的数据域存放英文单词，可用字符数组表示，单词重复出现时，链表中只保留一个，单词是否相等的判断使用strcmp函数，结点中增设计数域，统计单词重复出现的次数。

**typedef** **struct** node

{**int**  freg；∥频度域，记单词出现的次数。

**char** word[maxsize]；∥maxsize是单词中可能含有的最多字母个数。

**struct** node \*next；

}node, \*LinkedList;

(1)LinkedList creat（）

∥建立有n（n>0）个单词的单向链表，若单词重复出现，则只在链表中保留一个。

{LinkedList la；

la=（LinkedList）malloc（**sizeof**（node））；∥申请头结点。

la->next=null； ∥链表初始化。

**for**（i=1；i<=n；i++） ∥建立n个结点的链表

{scanf（“%s”，a）； ∥a是与链表中结点数据域同等长度的字符数组。

p=la->next；pre=p; ∥p是工作指针,pre是前驱指针。

**while**（p!=null）

**if**（strcmp(p->data，a）==0) {p->freg++；**break**；} ∥单词重复出现,频度增1。

**else** {pre=p；p=p->next；} ∥指针后移。

**if**（p==null） ∥该单词没出现过，应插入。

{p=（LinkedList）malloc（**sizeof**（node））；

strcopy（p->data，a）；p->freg=1；p->next=null；pre->next=p；

} ∥将新结点插入到链表最后。

}∥结束**for**循环。

**return**（la）；

}∥结束creat算法。

(2) **void** CreatOut（ ）

∥建立有n个单词的单向链表，重复单词只在链表中保留一个,最后输出频度最高的k个单词。

{LinkedList la；

la=（LinkedList）malloc（**sizeof**（node））；∥申请头结点。

la->next=null； ∥链表初始化。

**for**（i=1；i<=n；i++） ∥建立n个结点的链表

{scanf（“%s”，a）； ∥a是与链表中结点数据域同等长度的字符数组。

p=la->next；pre=p; ∥p是工作指针,pre是前驱指针。

**while**（p!=null）

**if**（strcmp(p->data，a)==0)

{p->freg++； ∥单词重复出现,频度增1。

pre->next=p->next; ∥先将p结点从链表上摘下，再按频度域值插入到合适位置

pre=la; q=la->next;

**while**(q->freg>p->freg) (pre=q; q=q->next; )

pre->next=p; p->next=q; ∥将p结点插入到合适位置

}

**else** {pre=p；p=p->next；} ∥指针后移。

**if**（p==null） ∥该单词没出现过，应插入到链表最后。

{p=（LinkedList）malloc（**sizeof**（node））；

strcopy（p->data，a）；p->freg=1；p->next=null；pre->next=p；

}∥**if** 新结点插入。

}∥结束**for**循环建表。

**int** k,i=0;

scanf(“输入要输出单词的个数%d”,&k);

p=la->next;

**while** (p && i<k) ∥输出频度最高的k个单词

{printf（“第%3d个单词%s出现%3d次\n”，++i，p->data，p->freg）；

p=p->next;

}

**if** (!p)

printf(“给出的%d值太大\n”,k);

}//结束算法

29．[题目分析] 双向循环链表自第二结点至表尾递增有序，要求将第一结点插入到链表中，使整个链表递增有序。由于已给条件（a1<x<an），故应先将第一结点从链表上摘下来，再将其插入到链表中相应位置。由于是双向链表，不必象单链表那样必须知道插入结点的前驱。

**void** DInsert（DLinkedList dl）

∥dl是无头结点的双向循环链表，自第二结点起递增有序。本算法将第一结点（a1<x<an）插入到链表中，使整个链表递增有序。

{s=la； ∥s暂存第一结点的指针。

p=la->next；p->prior=la->prior；p->prior->next=p；∥将第一结点从链表上摘下。

**while**（p->data<x）p=p->next； ∥查插入位置

s->next=p；s->prior=p->prior；p->prior->next=s；p->prior=s；∥插入原第一结点s

}∥算法结束。

[算法讨论] 由于题目已给a1<x<an，所以在查找第一结点插入位置时用的循环条件是p->data<x，即在a1和an间肯定能找到第一结点的插入位置。若无此条件，应先看第一结点数据域值x是否小于等于a1，如是，则不作任何操作。否则，查找其插入位置，循环控制要至多查找完a1到an结点。

**if**（p->data<x）p=p->next；**else** **break**；

30．[题目分析] 在顺序存储的线性表上删除元素，通常要涉及到一系列元素的移动（删第i个元素，第i+1至第n个元素要依次前移）。本题要求删除线性表中所有值为item的数据元素，并未要求元素间的相对位置不变。因此可以考虑设头尾两个指针（i=1，j=n），从两端向中间移动，凡遇到值item的数据元素时，直接将右端元素左移至值为item的数据元素位置。

**void** Delete（ElemType A[ ]，**int**  n）

∥A是有n个元素的一维数组，本算法删除A中所有值为item的元素。

{i=1；j=n；∥设置数组低、高端指针（下标）。

**while**（i<j）

{**while**（i<j && A[i]!=item）i++； ∥若值不为item，左移指针。

**if**（i<j）**while**（i<j && A[j]==item）j--；∥若右端元素值为item，指针左移

**if**（i<j）A[i++]=A[j--]；

}

[算法讨论] 因元素只扫描一趟，算法时间复杂度为O（n）。删除元素未使用其它辅助空间，最后线性表中的元素个数是j。若题目要求元素间相对顺序不变，请参见本章三、填空题25的算法。

31．[题目分析] 本题所用数据结构是静态双向链表，其结构定义为：

**typedef** **struct** node

{**char** data[maxsize]；∥用户姓名，maxsize是可能达到的用户名的最大长度。

**int**  Llink，Rlink；∥前向、后向链，其值为乘客数组下标值。

}unode；

unode user[max]；∥max是可能达到的最多客户数。

设av是可用数组空间的最小下标，当有客户要订票时，将其姓名写入该单元的data域，然后在静态链表中查找其插入位置。将该乘客姓名与链表中第一个乘客姓名比较，根据大于或小于第一个乘客姓名，而决定沿第一个乘客的右链或左链去继续查找，直到找到合适位置插入之。

**void** Insert（unode user[max],**int** av）

∥user是静态双向链表，表示飞机票订票系统，元素包含data、Llink和Rlink三个域，结点按来客姓名排序。本算法处理任一乘客订票申请。

{scanf（“%s”，s）； ∥s是字符数组，存放乘客姓名。

strcopy（user[av].data，s）；

p=1; ∥p为工作指针（下标）

**if**(strcmp（user[p].data，s）<0）∥沿右链查找

{**while** (p!=0 && strcmp（user[p].data，s）<0）{pre=p; p=user[p].Rlink; }

user[av].Rlink=p；user[av].Llink=pre； ∥将新乘客链入表中

user[pre].Rlink=av; user[p].Llink=av;

}

**else** ∥沿左右链查找

{**while** (p!=0 && strcmp（user[p].data，s）>0）{pre=p; p=user[p].Llink; }

user[av].Rlink=pre；user[av].Llink=p； ∥将新乘客链入表中

user[pre].Llink=av; user[p].Rlink=av;

}

}∥算法结束

[算法讨论] 本算法只讨论了乘客订票情况，未考虑乘客退票。也未考虑从空开始建立链表。增加乘客时也未考虑姓名相同者（实际系统姓名不能做主关键字）。完整系统应有（1）初始化，把整个数组空间初始化成双向静态链表，全部空间均是可利用空间。（2）申请空间。当有乘客购票时，要申请空间，直到无空间可用为止。（3）释放空间。当乘客退票时，将其空间收回。由于空间使用无优先级，故可将退票释放的空间作为下个可利用空间，链入可利用空间表中。

32．[题目分析]首先在双向链表中查找数据值为x的结点，查到后，将结点从链表上摘下，然后再顺结点的前驱链查找该结点的位置。

DLinkList locate(DLinkList L，ElemType x)

∥ L是带头结点的按访问频度递减的双向链表，本算法先查找数据x,查找成功时结点的访问频度域增1，最后将该结点按频度递减插入链表中适当位置。

{ DLinkList p=L->next,q; ∥p为L表的工作指针，q为p的前驱，用于查找插入位置。

**while** (p && p->data !=x) p=p->next; ∥ 查找值为x的结点。

**if** (!p) {printf(“不存在值为x的结点\n”); exit(0);}

**else** { p->freq++; ∥ 令元素值为x的结点的freq域加1 。

p->next->pred=p->pred; ∥ 将p结点从链表上摘下。

p->pred->next=p->next;

q=p->pred; ∥ 以下查找p结点的插入位置

**while** (q !=L && q->freq<p->freq) q=q->pred;

p->next=q->next; q->next->pred=p;∥ 将p结点插入

p->pred=q; q->next=p；

}

**return**(p); ∥返回值为x的结点的指针

} ∥ 算法结束

33．[题目分析] 题目要求按递增次序输出单链表中各结点的数据元素，并释放结点所占存储空间。应对链表进行遍历，在每趟遍历中查找出整个链表的最小值元素，输出并释放结点所占空间；再查次最小值元素，输出并释放空间，如此下去，直至链表为空，最后释放头结点所占存储空间。当然，删除结点一定要记住该结点的前驱结点的指针。

**void** MiniDelete（LinkedList head）

∥head是带头结点的单链表的头指针，本算法按递增顺序输出单链表中各结点的数据元素，并释放结点所占的存储空间。

{**while**（head->next!=null） ∥循环到仅剩头结点。

{pre=head； ∥pre为元素最小值结点的前驱结点的指针。

p=pre->next； ∥p为工作指针

**while**（p->next!=null）

{**if**（p->next->data<pre->next->data）pre=p；∥记住当前最小值结点的前驱

p=p->next；

}

printf（pre->next->data）；∥输出元素最小值结点的数据。

u=pre->next；pre->next=u->next; free（u）；∥删除元素值最小的结点，释放结点空间

}∥ **while**（head->next!=null）

free（head）；} ∥释放头结点。

[算法讨论] 算法中使用的指针变量只有pre，p和u三个，请读者细心体会。要注意没特别记最小值结点，而是记其前驱。

34．[题目分析] 留下三个链表中公共数据，首先查找两表A和B中公共数据，再去C中找有无该数据。要消除重复元素，应记住前驱，要求时间复杂度O（m+n+p），在查找每个链表时，指针不能回溯。

LinkedList Common（LinkedList A，B，C）

∥A，B和C是三个带头结点且结点元素值非递减排列的有序表。本算法使A表仅留下三个表均包含的结点，且结点值不重复，释放所有结点。

{pa=A->next；pb=B->next；pc=C->next；∥pa,pb和pc分别是A,B和C三个表的工作指针。

pre=A；∥pre是A表中当前结点的前驱结点的指针。

**while**（pa && pb && pc）∥当A,B和C表均不空时，查找三表共同元素

{ **while**（pa && pb）

**if**（pa->data<pb->data）{u=pa；pa=pa->next；free（u）；}∥结点元素值小时，后移指针。

**else** **if**（pa->data> pb->data）pb=pb->next；

**else** **if** (pa && pb) ∥处理A和B表元素值相等的结点

{**while**（pc && pc->data<pa->data）pc=pc->next；

**if**(pc)

{**if**（pc->data>pa->data）∥pc当前结点值与pa当前结点值不等,pa后移指针。

{u=pa；pa=pa->next；free（u）；}

**else** ∥pc，pa和pb对应结点元素值相等。

{**if**(pre==A){ pre->next=pa；pre=pa；pa=pa->next} ∥结果表中第一个结点。

**else** **if**(pre->data==pa->data) ∥（处理）重复结点不链入A表

{u=pa；pa=pa->next；free（u）；}

**else** {pre->next=pa；pre=pa；pa=pa->next；}∥将新结点链入A表。

pb=pb->next；pc=pc->next；∥链表的工作指针后移。

} }∥**else** pc，pa和pb对应结点元素值相等

**if**（pa==null）pre->next=null；∥原A表已到尾,置新A表表尾

**else** ∥处理原A表未到尾而B或C到尾的情况

{pre->next=null； ∥置A表表尾标记

**while**（pa!=null） ∥删除原A表剩余元素。

{u=pa；pa=pa->next；free（u）；}

[算法讨论] 算法中A表、B表和C表均从头到尾（严格说B、C中最多一个到尾）遍历一遍，算法时间复杂度符合O（m+n+p）。算法主要由**while**（pa && pb && pc）控制。三表有一个到尾则结束循环。算法中查到A表与B表和C表的公共元素后，又分三种情况处理：一是三表中第一个公共元素值相等的结点；第二种情况是，尽管不是第一结点，但与前驱结点元素值相同，不能成为结果表中的结点；第三种情况是新结点与前驱结点元素值不同，应链入结果表中，前驱指针也移至当前结点，以便与以后元素值相同的公共结点进行比较。算法最后要给新A表置结尾标记，同时若原A表没到尾，还应释放剩余结点所占的存储空间。