跨考教育
www.kuakao.com

跨考教育计算机教研室 编

全面贯彻“跨越135分”辅导理念

全国硕士研究生入学统一考试

计算机核心习题集

QUANGUO SHUOSHI YANJIUSHENG RUXUE TONGYI KAOSHI
JISUANJI HEXIN XITIJI LVBAOSHU

绿宝书

★核心习题精解与核心练习　★答题技巧
★重点、难点、考点梳理　★备考说明
★真题演练及答案解析　★自我检测

北京邮电大学出版社
www.buptpress.com

最新版

全国硕士研究生入学统一考试
计算机核心习题集·绿宝书

跨考教育计算机教研室　编

北京邮电大学出版社
·北京·

内 容 简 介

“计算机考研跨越135分必备”系列包括四个分册：(1)《全国硕士研究生入学统一考试计算机基础综合辅导·蓝宝书》；(2)《全国硕士研究生入学统一考试计算机考研核心习题集·绿宝书》；(3)《全国硕士研究生入学统一考试计算机历年真题全真解析·黄宝书》；(4)《全国硕士研究生入学统一考试计算机全真模拟题及答案·红宝书》。每一个分册的编写都凝聚了跨考教育教授们多年的研究心血。

本书主要主要由数据结构、计算机组成原理、操作系统及计算机网络四部分组成。对于每一部分，我们都按大纲的要求，在每一章背后附带配套习题。其中，习题类型分为两类，即核心习题精解和核心习题练习。核心习题精解部分给出问题的详细说明并予以适当拓展；核心习题练习要求学员尽量独立完成，我们仅给出了参考答案。重点章节每章约有20道习题，非重点章节约为10道，核心习题精解与核心习题练习所占比例约为4∶6。

本书不仅特别适合在硕士研究生入学考试中参加理工类科目考试的考生，也适合各大院校学习理工类高级课程的师生，对于参加高级职称考试及其他相关专业人员来说，本书也是一本宝贵的学习和了解计算机课程的参考资料。

图书在版编目(CIP)数据

全国硕士研究生入学统一考试计算机核心习题集：绿宝书/跨考教育计算机教研室编. --北京：北京邮电大学出版社，2011.7

ISBN 978-7-5635-2652-9

Ⅰ.①全… Ⅱ.①跨… Ⅲ.①电子计算机—研究生—入学考试—习题集 Ⅳ.①TP3-44

中国版本图书馆CIP数据核字(2011)第118862号

书　　名：全国硕士研究生入学统一考试计算机核心习题集·绿宝书
作　　者：跨考教育计算机教研室
责任编辑：满志文
出版发行：北京邮电大学出版社
社　　址：北京市海淀区西土城路10号(邮编：100876)
发 行 部：电话：010-62282185　传真：010-62283578
E-mail：publish@bupt.edu.cn
经　　销：各地新华书店
印　　刷：北京忠信诚胶印厂
开　　本：787 mm×1 092 mm　1/16
印　　张：19.5
字　　数：486千字
版　　次：2011年7月第1版　2011年7月第1次印刷

ISBN 978-7-5635-2652-9　　定　价：38.50元

前　言

本书是根据教育部考试中心公布的《计算机学科专业基础综合考试大纲》的要求，特别是突出大纲中变化部分的内容，对习题进行命制和筛选的，力求习题能够切中考试要害。

目前市面上的考研习题集多得数不胜数，但能够有效使学员在花费相对少时间掌握具有代表性的习题的同时，最大程度上提升解题能力的却少之又少。本着这个原则，我们竭尽所能命制和筛选出具有代表性的习题，以期学员能够在尽量短的时间内掌握更多的内容。在习题命制和筛选过程中，对于难度较大的题目，即涉及知识点相对较多且需要考生融会贯通的试题，编者用通俗易懂的语言进行深入的解析，以使学员能够明晰试题涉及的知识点，引导学员举一反三，达到高效的训练目的。

本书主要由数据结构、计算机组成原理、操作系统及计算机网络四部分组成。对于每一部分，我们都按大纲的要求，在每一章背后附带配套习题。其中，习题类型分为两类，即核心习题精解和核心习题练习。核心习题精解部分给出问题的详细说明并予以适当拓展；核心习题练习要求学员尽量独立完成，我们仅给出答案。重点章节每章约有 20 道习题，非重点章节约为 10 道，核心习题精解与核心习题练习所占比例约为 4∶6。

本书由全国计算机专业排名领先的清华大学、北京邮电大学、国防科技大学等名校的资深教授、专家和一线教学骨干等组成的强大作者队伍精心打造而成，在此对他们严谨的治学态度和付出的智慧与努力表示感谢！

编者在多年教学经验和考研试题命制研究的基础上，总结了大量前人的经验，并不断改进、创新，力争使本书成为一个新的高点。不过，由于时间仓促，本书难免会存在一些错误和遗漏，恳请各位考生朋友给予批评和指正，不胜感激！您的任何疑问，可以在跨考考研论坛（http://bbs.kuakao.com/）上发布，我们会第一时间回答您的疑问。也可通过以下邮箱与我们联系：bjbaba@263.net。

预祝广大考生梦圆未来！

跨考教育计算机教研室于北京

目　录

第四部分 计算机网络

数据结构

第1章

线性表

习题精解 数据结构

1.1 核心习题精解

核心习题 01 有两个按元素值递增次序排列的线性表，均以单链表形式存储。请编写算法将这两个单链表归并为一个按元素值递减次序排列的单链表，并要求利用原来两个单链表的结点存放归并后的单链表。

解答：[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(LinkedList la,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,只能两者取一,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。

与本题类似的其他题解答如下:

(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。

核心习题 02　带头结点且头指针为 ha 和 hb 的两线性表 A 和 B 分别表示两个集合。两表中的元素皆为递增有序。请写一算法求 A 和 B 的并集 $A \cup B$。要求该并集中的元素仍保持递增有序。且要利用 A 和 B 的原有结点空间。

解答：[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（$A \cup B$，$A \cap B$，$A-B$）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
```

与本题类似的其他几个题解答如下:

(1) 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
```

(2) 本题基本与(1)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。

(3) 本题与上面(3)算法相同,只是结果表要另辟空间。

(4)[**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 3 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，不这样做，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。

核心习题 03 已知L1、L2分别为两循环单链表的头结点指针,m,n分别为L1、L2表中数据结点个数。要求设计一算法,用最快速度将两表合并成一个带头结点的循环单链表。

解答:[题目分析] 循环单链表L1和L2数据结点个数分别为m和n,将两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
```

类似本题叙述的其他题解答如下:

(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
```

核心习题 04 顺序结构线性表 LA 与 LB 的结点关键字为整数。LA 与 LB 的元素按非递减有序,线性表空间足够大。试用类 PASCAL 语言给出一种高效算法,将 LB 中元素合到 LA 中,使新的 LA 的元素仍保持非递减有序。高效指最大限度的避免移动元素。

解答:[**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(副本)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,"最大限度地避免移动元素",是"一种高效算法"。

核心习题 05 已知不带头结点的线性链表list,链表中结点构造为(data、link),其中data为数据域,link为指针域。请写一算法,将该链表按结点数据域的值的大小从小到大重新链接。要求链接过程中不得使用除该链表以外的任何链结点空间。

解答:[**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;
  }
```

核心习题 06 设 L 为单链表的头结点地址，其数据结点的数据都是正整数且无相同的，试设计利用直接插入的原则把该链表整理成数据递增的有序单链表的算法。

解答：[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始，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;
}
```

核心习题 07 设 Listhead 为一单链表的头指针，单链表的每个结点由一个整数域 DATA 和指针域 NEXT 组成，整数在单链表中是无序的。编一个 PASCAL 过程，将 Listhead 链中结点分成一个奇数链和一个偶数链，分别由 P、Q 指向，每个链中的数据按由小到大排列。程序中不得使用 NEW 过程申请空间。

解答：[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
```

解答:[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。

类似本题的其他题解答如下:

(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 语言程序书写如下:

```
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）。

（2）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为链表中结点的序号
   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
```