2010年
全国硕士研究生入学统一考试

MASTER DEGREE

QUANGUO SHUOSHI YANJIUSHENG RUXUE TONGYI KAO

主编 邵增珍 刘永华

计算机 专业基础综合

TIXINGLIANXI YU QUANZHENMONI

题型练习与全真模拟

JISUANJI ZHUANYE JICHU ZONGHE

紧扣教育部考试中心
2010年全国硕士研究生入学统一考试
计算机专业基础综合考试大纲编写

最新版

全国硕士研究生入学统一考试

计算机专业基础综合题型练习与全真模拟

主　编　邵增珍　刘永华

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专业基础综合题型练习与全真模拟/邵增珍,刘永华主编.—济南:山东人民出版社,2009.10
ISBN 978-7-209-05065-4

Ⅰ.全... Ⅱ.①邵...②刘... Ⅲ.电子计算机-研究生-入学考试-习题 Ⅳ.TP3-44

中国版本图书馆 CIP 数据核字(2009)第 184941 号

责任编辑:王金凤 隋小山
封面设计:武 斌

全国硕士研究生入学统一考试
计算机专业基础综合题型练习与全真模拟
邵增珍 刘永华 主编

山东出版集团
山东人民出版社出版发行
社 址:济南市经九路胜利大街 39 号 邮 编:250001
网 址:http://www.sd-book.com.cn
发行部:(0531)82098027 82098028
新华书店经销
日照报业印刷有限公司印装
规 格 16 开 (184mm×260mm)
印 张 26.25
字 数 640 千字
版 次 2009 年 10 月第 1 版
印 次 2009 年 10 月第 1 次
ISBN 978-7-209-05065-4
定 价 45.00 元

INSTRUCTION

编写说明

许多考生在进行了一轮复习后，多少感到有些迷茫，主要是不了解自己的复习到底达到了什么程度？离考研的要求还有多远？还有哪些知识需要掌握？要解决这些问题，做一遍严格按照考试大纲编写的题型练习和模拟试题是最好的方法。

经典的题型练习和模拟试题，是命题专家认真研究分析教育部考试中心公布的最新考试大纲、试卷结构和分数比值后形成的，既反映了考试大纲的基本要求，又蕴涵着命题的基本思想和发展趋势，是广大考生了解全国硕士研究生入学考试最直接的第一手资料，考生从中可直观地了解到硕士研究生入学考试的试题类型、考点分布和难易程度。

有人说："做一遍好的题型练习和模拟试题，考研就成功了一半。"这是至理名言。因为经典的题型练习和模拟试题最直接、最全面地显现着考试大纲的基本原则。这也是广大考生一向对题型练习和模拟试题重视的原因所在。从这个角度讲，做一遍题型练习和模拟试题，离考研成功就不远了。

"题型练习和模拟试题"的构成最大限度地体现了考试大纲的基本精神，是检验考生对考试大纲理解和对基础知识掌握的标尺。考生对基础知识进行了一轮复习后，做一遍题型练习和模拟试题是对自己最好的检验。既能从中找到考研的信心，又能找出自己的不足之所在，拾遗补缺，使以后的复习更有目的和针对性，做到心中有数，了然于胸。做一遍题型练习，本身就是一次很大的收获。

本书由专家对"题型练习和模拟试题"进行了解答，考生可从中看到解答问题的方法和规范，开阔解题思路，增强答题技巧，提高应试水平，最大限度地发挥自己的水平。有许多过来的考生反映，该看的教材都看了，辅导书也读了不少，自认为对基础知识掌握得比较好，却考不出好成绩。这其中一个重要的原因就是答题技巧和应试水平的欠缺。通过做一遍质量可靠的题型练习和模拟试题，可从根本上解决这一问题，达到事半功倍的效果，用最少的时间，取得最好的成绩。基于以上认识，我们编写了全国硕士研究生入学考试题型练习和模拟试题，以期对广大考生有所帮助。

为帮助考生更好地理解考试大纲和把握命题方向，本书严格按照教育部考试中心公布的考试大纲、试卷结构和考分比值进行编写，供广大考生参考使用。希望考生能从中提取精华，受到启示，获得收益。达到举一反三、触类旁通的效果。

为了方便考生复习，书中对全部的题型练习都作了参考答案。制作答案时参考了多部经典教材和教学参考书，由于体例的原因未能在解析时一一注明，在此对教材和教学参考书的编写者表示衷心的感谢。

本书由邵增珍、刘永华主编，由柏静、陈莉、李少辉、徐功文副主篇。组成原理部分由

柏静编写，操作系统部分由陈莉编写，数据结构部分由李少辉编写，网络系统部分由刘永华编写。他们多是长期从事研究生教学和管理的一线教师和对考研有亲身体验的博士、硕士，有丰富的出题经验和应试经验。由于时间仓促，可能会有这样那样的问题，希望读者批评指正，以便再版时修订改正。

CONTENTS

计算机专业基础综合题型练习与全真模拟

目 录

第一部分　数据结构

第一章　线性表

考试要求

* 线性表的定义和基本操作
* 线性表的实现
  * 顺序存储结构
  * 链式存储结构
  * 线性表的应用

题型练习

一、单项选择题

【1－1】线性表是(　　)。

A. 一个有限序列，可以为空　　B. 一个有限序列，不可以为空

C. 一个无限序列，可以为空　　D. 一个无限序列，不可以为空

【1－2】在一个长度为 n 的顺序表中删除第 i 个元素（0<＝i<＝n）时，需向前移动(　　)个元素。

A. n－i　　B. n－i+1　　C. n－i－1　　D. i

【1－3】在一个长度为 n 的顺序表中向第 i 个元素（0< i<n+1）之前插入一个新元素时，需向后移动(　　)个元素。

A. n－i　　B. n－i+1　　C. n－i－1　　D. i

【1－4】下述哪一条是顺序存储结构的优点？(　　)

A. 存储密度大　　B. 插入运算方便

C. 删除运算方便　　D. 可方便地用于各种逻辑结构的存储表示

【1－5】下面关于线性表的叙述中，错误的是哪一个？(　　)

A. 线性表采用顺序存储，必须占用一片连续的存储单元

B. 线性表采用顺序存储，便于进行插入和删除操作

C. 线性表采用链接存储，不必占用一片连续的存储单元

D. 线性表采用链接存储，便于插入和删除操作

【1－6】若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算，则利用(　　)存储方式最节省时间。

A. 顺序表　　B. 双链表

C. 带头结点的双循环链表　　D. 单循环链表

【1－7】对于顺序存储的线性表，访问结点和增加、删除结点的时间复杂度为(　　)。

A. O(n),O(n)　B. O(n),O(1)　　C. O(1), O(n)　　D. O(1), O(1)

【1－8】(1) 静态链表既有顺序存储的优点，又有动态链表的优点，所以，它存取表中第 i 个元素的时间与 i 无关。(2) 静态链表中能容纳的元素个数的最大数在表定义时就确定了，以后不能增加。(3) 静态链表与动态链表在元素的插入、删除上类似，不需做元素的移动。以上错误的是(　　)。

A. (1)，(2)　　B. (1)　　C. (1)，(2)，(3)　　D. (2)

【1－9】某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，则采用(　　)存储方式最节省运算时间。

A. 单链表　　B. 仅有头指针的单循环链表

C. 双链表　　D. 仅有尾指针的单循环链表

【1－10】在双向循环链表中，在 p 所指的结点之后插入 s 指针所指的结点，其操作是(　　)。

A. p－>next=s; s－>prior=p; p－>next－>prior=s; s－>next=p－>next;

B. s－>prior=p; s－>next=p－>next; p－>next=s; p－>next－>prior=s;

C. p－>next=s; p－>next－>prior=s; s－>prior=p; s－>next=p－>next;

D. s－>prior=p; s－>next=p－>next; p－>next－>prior=s; p－>next=s;

【1－11】若长度为 n 的线性表采用顺序存储结构，在其第 i 个位置插入一个新元素的算法的时间复杂度为(　　)(1<=i<=n+1)。

A. O(0)　　B. O(1)　　C. O(n)　　D. $O(n^2)$

【1－12】设单链表中结点的结构为

```
typedef struct node {  //链表结点定义
ElemType data;   //数据
struct node * Link;   //结点后继指针
} ListNode;
```

已知指针 p 所指结点不是尾结点，若在 p 之后插入结点 s，则应执行下列哪一个操作？(　　)

A. s－>link = p; p－>link = s;

B. s－>link = p－>link; p－>link = s;

C. s－>link = p－>link; p = s;

D. p－>link = s; s－>link = p;

二、综合应用题

【1－13】设计在无头结点的单链表中删除第 i 个结点的算法。

【1－14】设计将带表头的链表逆置算法。

【1－15】设顺序表中的数据元素递增有序，编写一算法将元素 x 插入到顺序表的适当位置上，并保证该表的有序性。

【1－16】设线性表 A=（a_1，a_2，a_3，…，a_n）以带头结点的单链表作为存储结构。编写一个函数，对 A 进行调整，使得当 n 为奇数时 A=（a_2，a_4，…，a_{n-1}，a_1，a_3，…，

a_n），当 n 为偶数时 A=（a_2，a_4，…，a_n，a_1，a_3，…，a_{n-1}）。

【1－17】试分别用顺序表和单链表作为存储结构，实现将线性表（a_0，a_1，a_2，……，a_{n-1}）就地逆置的操作，所谓“就地”，是指辅助空间应为 O(1)。

【1－18】假设以带头结点的单链表表示有序表，单链表的类型定义如下：

```
typedef struct node {
  DataType data;
  struct node * next
} LinkNode, * LinkList;
```

编写算法，从有序表 A 中删除所有和有序表 B 中元素相同的结点。

【1－19】设带表头结点的双向链表的定义为

```
typedef int ElemType;
typedef struct dnode { //双向链表结点定义
ElemType data; //数据
struct dnode * lLink, * rLink; //结点前驱与后继指针
} DblNode;
typedef DblNode * DblList; //双向链表
```

试设计一个算法，改造一个带表头结点的双向链表，所有结点的原有次序保持在各个结点的右链域 rLink 中，并利用左链域 lLink 把所有结点按照其值从小到大的顺序连接起来。

【1－20】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。

参考答案

一、单项选择题

【1－1】A。线性表是由相同类型的结点组成的有限序列。如：由 n 个结点组成的线性表

$$(a_1, a_2, \cdots, a_n)$$

a_1 是最前结点，a_n 是最后结点。结点也称为数据元素或者记录。

线性表中结点的个数称为其长度。长度为 0 的线性表称为空表。

所以答案为 A。

【1－2】A。一般情况下，删除顺序表的第 i（1<=i<=n）个元素时需要将第 i+1 到第 n 个元素（共 n－i 个元素）依次向前移动一个位置。所以答案为 A。

【1－3】B。一般情况下，在顺序表的第 i（1<=i<=n）个元素之前插入一个元素，需要将第 n 至 i 的元素（共 n－i+1 个元素）向后移动一个位置。所以答案为 B。

【1－4】A。顺序存储利用物理的邻接关系表示数据元素之间的逻辑关系，因此没有必要设置指针域，所以其存储密度比链式存储大，但是插入运算和删除运算都需大量移动数据元素，并不方便；D 选项并不是顺序存储结构的优点。所以答案为 A。

【1－5】B。线性表采用顺序存储，并不便于进行插入和删除操作。

【1－6】A。“存取任一指定序号”最好的方法是实现“随机存取”，则可采用顺序表。并且，因为插入和删除操作都是在最后进行的，所以无需大量移动数据元素，选项 A 是最

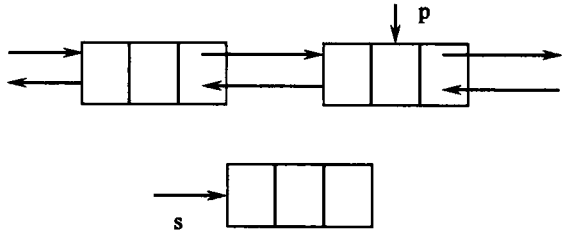
合适的。

【1－7】C。顺序存储可以实现“随机存取”，因此访问结点的时间复杂度为 O(1)，而插入、删除结点由于涉及到大量移动元素，故其时间复杂度为 O(n)。

【1－8】B。静态链表使用结构体数组来实现线性链表的功能。因为其用游标 cur 来指示下一个数据元素的存储位置，所以存取数据时静态链表同线性链表（单链表）是相似的。也就是说，静态链表在存取表中第 i 个元素的时间同 i 是相关的。

【1－9】D。仅有尾指针的单循环链表，可以非常方便地找到尾结点，尾结点后面的第一个结点往往是头结点，头结点的下一个结点就是第线性表的第一个结点。对最后一个元素和第一个元素操作对带尾指针的单循环链表是非常方便的。

【1－10】D。同单链表相比，双向链表的插入、删除操作需同时修改两个指针。



步骤：s－>prior = p－>prior;

p－>prior－>next = s;

s－>next = p;

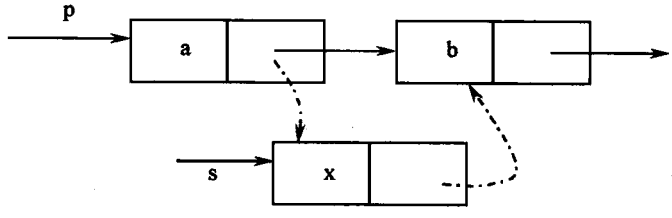
p－>prior = s;

【1－11】C。顺序存储的线性表在插入新元素时，需要将第 i 个元素到第 n 个元素均向后移动一个单位，插入的新元素成为第 i 个元素，原来的第 i 个元素成为第 i+1 个元素，原来的第 n 个元素成为第 n+1 个元素，线性表的长度加 1，插入操作的主要工作都放在移动元素上。假设线性表中含有 n 个数据元素，在进行插入操作时，若假定在 n+1 个位置上插入元素的可能性均等，则平均移动元素的个数为：

$$E_iS = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{1}{2}$$

除非插入在线性表的最后，否则都要移动元素。所以其时间复杂度为 O(n)，答案为 C。

【1－12】B。



二、综合应用题

【1－13】算法思想为：

(1) 应判断删除位置的合法性，当 i<0 或 i>n－1 时，不允许进行删除操作；

(2) 当 i=0 时，删除第一个结点；

(3) 当 0<i<n 时，允许进行删除操作，但在查找被删除结点时，须用指针记住该结点的前趋结点。算法描述如下：

```
delete (LinkList *q, int i)
{  //在无头结点的单链表中删除第 i 个结点
    LinkList  *p, *s;
    int  j;
    if (i<0)
      printf ( "Can't delete");
    else if (i= =0)
       {  s=q;
          q=q->next;
          free (s);
       }
    else
       {  j=0; s=q;
          while ( (j<i) && (s! = NULL))
           {  p=s;
              s=s->next;
              j++;
           }
          if (s= =NULL)
            printf ( "Can't delete");
          else
             {  p->next=s->next;
                free (s);
             }
       }
}
```

【1－14】解析：设单循环链表的头指针为 head，类型为 LinkList。逆置时需将每一个结点的指针域做一修改，使其原前趋结点成为后继。如要更改 q 结点的指针域时，设 s 指向其原前趋结点，p 指向其原后继结点，则只需进行 q->next=s；操作即可，算法描述如下：

```
void invert (LinkList *head)
{  //逆置 head 指针所指向的单循环链表
   linklist *p, *q, *s;
   q=head;
```

```
    p=head->next;
    while (p! =head)    //当表不为空时，逐个结点逆置
    {  s=q;
       q=p;
       p=p->next;
       q->next=s;
    }
    p->next=q;
}
```

【1-15】只要从终端结点开始往前找到第一个比 x 小（或相等）的结点数据，在这个位置插入就可以了。算法描述如下：

```
SqList * Insert _ SqList (SqList * L, int i, Elemtype x)
{
    if ( (i<1) || (i>L->length+1))
     {
      printf ("插入位置不合理"); exit (1);
     }
if (L->length>=L->MaxSize-1)
{
    printf ("顺序表已满，不能再插入!"); exit (1);
}
for (m=L->length-1; m>=i-1; --m)
    L->data [m+1] =L->data [m];
L->data [i-1] =x;
L->length++;
return L;
}
```

【1-16】

方法一：

```
void f (LinkList L)
{
    LinkList p, q, r;
    p=L; //*p 是偶数尾结点
    q=L->next; //*q 是当前奇数结点
    while (q&&q->next) //尚有未调整的偶数结点
     {
      r=q->next; //r 指向当前偶数结点
      q->next=r->next; //删除偶数结点
      r->next=p->next;
      p->next=r;    //插入偶数结点
```

```
        p=p->next;
        q=q->next;
      }
}
```

方法二：

```
void f (LinkList L)
{
    LinkList p, h, r, s;
    if (! L->next || ! L->next->next) return;
    h=L->next;
    r=h; //h 和 r 分别为奇数结点的头、尾指针
    L->next=h->next;
    p=L->next; //p 指向偶数结点
    while (p->next) //2 个结点以上
     {
      s=p->next; //s 指向奇数结点
      p->next=s->next; //删除奇数结点
      r->next=s; //链接奇数结点
      r=s;
      if (p->next)
        p=p->next; //尚有偶数结点存在
     }
    p->next=h;
    r->next=NULL;
}
```

方法三：

```
void f (LinkList L)
{
    LinkList p, h, r1, r2, s;
    if (! L->next || ! L->next->next) return;
    h=L->next; //h 是奇数结点头指针
    r2=h; //r2 是奇数尾结点指针
    r1=L; //r1 是偶数结点指针
    p=h->next; //p 指向第 1 个偶数结点
    while (p)
     {
        s=p->next; //s 指向下一个奇数结点
        r1->next=p;
        r1=p; //链接偶数结点
        if (s)
```

```
        {
          p=s->next; // p 指向下一个偶数结点
          r2->next=s;
          r2=s; //链接奇数结点
         }
        else p=s; //偶数个数情况
     }
    r1->next=h;
    r2->next=NULL;
}
```

【1-17】

方法一：用顺序表作存储结构

```
struct SqList/
{
    ElemType *elem; // 存储空间基址
    int length; // 当前长度
    ……
     };
void InvertSqList (SqList &L)
   {
    int i; ElemType temp;
      for (i=0; i<L. length/2; i++)
     {
      temp=L. elem;
      L. elem=L. elem[L. length-i-1];
      L. elem[L. length-i-1]= temp;
     }
}
```

方法二：用顺序表作存储结构

```
void InvertSqList(SqList &L)6 R
{
    int i, j; ElemType temp;
    i=0; j= L. length-1;
      while (i<j)
       {
          temp=L. elem;
          L. elem=L. elem [j]
          L. elem[j]= temp
          i++; j--;
       }
```

```
}
```

方法三：用带头结点的单链表作存储结构

```
struct LNode
{
      ElemType data;
      LNode * next;
  };
  typedef LNode * LinkList;
Void InvertLinkList (LinkList &L)
   {
      p=L; L=NULL;
      while (p) {
                s=p; p=p->next;
                s->next=L;
    }
}
```

【1-18】

```
void deleted _ list(LinkList A, LinkList B)
{
    LinkList p, q, r;
    r=A;
    p=A->next;
    q=B->next;
    while (p! =NULL&&q! =NULL)
     {
       if (p->data>q->data)
           q=q->next
       else if (p->data<q->data)
          {
           r=p;
          p=p->next;
           }
         else
             {
             r->next=p->next;
             free(p);
             p=r->next;
             }
      }
}
```