21世纪高等学校计算机科学与技术专业规划教材

数据结构上机实验与习题解析

王成端　徐翠霞　主编

中国电力出版社
www.infopower.com.cn

21世纪高等学校计算机科学与技术专业规划教材

数据结构上机实验与习题解析

王成端　徐翠霞　　主　编
王凌云　赵家玲　张兴科　副主编

中国电力出版社
www.infopower.com.cn

内容提要

本书习题对应于教科书中的知识点，结合每一章的学习要求进行编排，全书内容包括实验指导、习题解析、模拟试题及参考答案。书中的每一个实验案例都有鲜明的主题和具体的要求，并给出可执行的源代码程序。另外，还在附录给出了习题和实验中涉及的常用数据结构的类型定义。

本书内容丰富、讲解通俗易懂，非常适合于高职高专计算机及相关专业数据结构课程的教学辅导参考书，同时也可作为广大工程技术人员和自学读者的辅助教材。

图书在版编目（CIP）数据

数据结构上机实验与习题解析 / 王成端，徐翠霞主编. —北京：中国电力出版社，2006

21 世纪高等学校计算机科学与技术专业规划教材

ISBN 7-5083-4281-X

Ⅰ.数... Ⅱ.①王...②徐... Ⅲ.数据结构－高等学校－教学参考资料 Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字（2006）第 047158 号

丛 书 名：21 世纪高等学校计算机科学与技术专业规划教材

书　　名：数据结构上机实验与习题解析

出版发行：中国电力出版社

地　　址：北京市三里河路 6 号　　邮政编码：100044

电　　话：（010）68362602　　传　　真：（010）68316497，88383619

本书如有印装质量问题，我社负责退换

服务电话：（010）88515918（总机）　　传　　真：（010）88518169

E-mail：infopower@cepp.com.cn

印　　刷：北京同江印刷厂

开本尺寸：185×260　　**印　　张**：12.5　　**字　　数**：310 千字

书　　号：ISBN 7-5083-4281-X

版　　次：2006 年 6 月北京第 1 版

印　　次：2006 年 6 月第 1 次印刷

印　　数：0001—4000 册

定　　价：19.00 元

前　言

数据结构是计算机专业的必修、主干课程之一，它旨在使读者学会分析研究数据对象的特性，学会数据的组织方法，以便选择合适的数据逻辑结构和存储结构以及相应的运算（操作），把现实世界中的问题转化为计算机内部的表示和处理，这是一个良好的程序设计技能训练过程。在整个教学或学习的过程中，解题能力和技巧的训练是一个重要的环节。为了帮助教师讲授“数据结构”，也为了帮助和指导读者更好地学习数据结构这门课程，我们根据多年来在教学一线和专升本考试辅导工作中积累的丰富经验，编写了这本适应于学生需求的《数据结构上机实验与习题解析》实训教材。

实践证明，理解课程内容与较好地解决实际问题之间存在着明显的差距，而算法设计完成的质量与基本的程序设计素质的培养是密切相关的。要想理解和巩固所学的基本概念、原理和方法，牢固地掌握所学的基本知识、基本技能，达到融会贯通、举一反三的目的，就必须多做、多练、多见。正是为了达到上述目的，书中用一些实际的应用，对一些重要的数据结构和算法进行解读。经过循序渐进地训练，就可以使读者掌握更多的程序设计技巧和方法，提高分析问题和解决问题的能力。

全书共包括三大部分：

第 1 部分是学习本课程应进行的实验。考虑到教学的要求，共安排了 8 个实验项目。为了提高学生分析问题和解决问题的能力，每个实验都选择了 2 个或 3 个案例，这些设计内容丰富、涉及面广、难易适当，能给学习数据结构这门课程的读者以启发，达到让读者掌握相关知识和开阔视野的目的。因此，对书中实验案例题目都作了解析，并给出了参考算法和源程序代码。

第 2 部分是根据教程中的习题而编写的答案以及相关知识的讲解。在解题过程中，根据题目的难易程度进行了不同的处理，对典型的有代表性的题目进行了解析，对简单易懂的题目则不加分析直接给出答案，而对有些题目则以解析的形式加以说明。

第 3 部分是针对教材所讲知识点的配套试题。试题内容全面，难度适中，既有试卷又有参考答案。

本书由王成端、徐翠霞担任主编，王凌云、赵家玲、张兴科担任副主编，陈永平、吴葳葳、石洋、陈姗等参与编写。其中第 1 部分的实验 1、实验 5，第 2 部分的第 1、2、6 章由王成端编写；第 1 部分的实验 2、实验 3，第 2 部分的第 3、4 章，第 3 部分的模拟试题及参考答案由徐翠霞编写；第 1 部分的实验 7，第 2 部分的第 8 章由王凌云编写；第 1 部分的实验 6，第 2 部分的第 7 章由赵家玲编写；第 1 部分的实验 4、实验 8，第 2 部分的第 5、9 章由张兴科编写。全书由王成端负责统稿。

本书习题丰富，讲解通俗易懂；程序设计观点新颖，实验面向测试、调试，具有启发性；可作为高等院校应用型本科计算机专业及相关专业数据结构课程的教学辅导书。

由于作者水平所限，书中错误之处在所难免，敬请广大读者批评指正。

作　者

2006 年 4 月

目　录

第 1 部分　实验指导

第 2 部分 习题解析

第1部分　实验指导

实验1　线　性　表

1.1　实验基础知识

1. 顺序表的特点及基本操作

线性表的顺序表表示，其特点是用物理位置上的邻接关系来表示结点间的逻辑关系，这一特点使得顺序表有以下的优缺点。

其优点是：

（1）无需为表示结点间的逻辑关系而增加额外的存储空间。

（2）可以方便地随机存取表中任一结点。

其缺点是：

（1）插入或删除运算不方便，除表尾的位置外，在表的其他位置上进行插入或删除操作都必须移动大量的结点，其效率较低。

（2）由于顺序表要求占用连续的存储空间，存储分配只能预先进行（静态分配）。因此，当表长变化较大时，难以确定合适的存储规模。若按可能达到的最大长度预先分配表空间，则可能造成一部分空间长期闲置而得不到充分利用；若事先对表长估计不足，则插入操作可能使表长超过预先分配的空间而造成溢出。

2. 链表的特点及基本操作

为了克服顺序表的缺点，可以采用链接方式存储线性表。值得指出的是，链接存储是最常用的存储方法之一，它不仅可用来表示线性表，而且可以用来表示各种非线性的数据结构。

链表是一种动态存储的数据结构，在创建链表时利用动态存储分配函数申请存储空间。链表的每个结点用一个结构体变量表示。每个结点包含数据域和指针域。指针域用来指向该结点的下一个结点。每个链表有一个头指针，存放链表的首地址。链表的最后一个结点其指针域的值为NULL，表示链表结束。链表可以进行建立、输出、结点的插入与删除等基本操作。

在顺序表中，我们是用一组地址连续的存储单元来依次存放线性表的结点，因此结点的逻辑次序和物理次序一致。而链表则不然，链表是用一组任意的存储单元来存放线性表的结点，这组存储单元既可以是连续的，也可以是不连续的，甚至是零散分布在内存中的任何位置上。因此，链表中结点的逻辑次序和物理次序不一定相同。

1.2　实验案例及其分析

1. 约瑟夫（Joseph）问题

【案例描述】

编号为1，…，n的n个人按顺时针方向围坐一圈，从第1号的人开始按顺时针方向自1开始顺序报数，报到m时停止报数（m<n）。报m的人出列，从他在顺时针方向上的下一个人开始重新从1报数，如此下去，直至所有人全部出列为止。

【案例要求】

试设计一个程序模拟约瑟夫问题，按照出列的顺序打印出各人的编号。

（1）建立一个顺序表或循环单链表存储n个人的编号信息。

（2）在（1）中建立的顺序表或循环单链表上实现约瑟夫问题的求解。

【数据结构】

```
typedef  struct point
{  int data;
   int  No;
   struct point *next;
}LNode,*LinkList;
```

【案例分析】

（1）设置变量num标记出列人数，控制循环执行次数，当num取值为0～n-1时，执行循环，否则退出循环。

（2）设置变量count累计报数人数，决定某一个人是否出列，count从1开始累计，当count=m时，做如下处理：

①刚刚报数的人需要出列，即将其所在结点从表中删除。若采用循环链表结构，为了实现该操作，需要两个工作指针pre和p，分别指向链表中的两个相邻结点；

②出列人数num加1；

③count重新置1，准备开始下一轮报数。

【案例实现】

```
#include "stdlib.h"
#include "stdio.h"
#define NULL 0
typedef  struct point
{ int data;
  int  No;
  struct point *next;
}LNode,*LinkList;
int n,m;
LinkList  create()       /*生成循环单链表并返回*/
{int i;
 LinkList  head, tail, new;
 head=NULL;
 printf("\ninput n  (总人数): ");
 scanf("%d",&n);
 printf("\ninput  m : ");
 scanf("%d", &m);
 for (i=1;i<=n;i++)
  { new=(LinkList)malloc(sizeof(LNode));
    new->No=i;
    if (head==NULL)      /*链表的第一个结点插入*/
```

```
        { head=new;      tail=head; }
     else                  /*链表的其余结点插入*/
        { tail->next=new;   tail=new;}
   }
  tail->next=head;        /*循环单链表的生成*/
  return  head;
}
 void search(LinkList  head )
 {                        /*用循环链表实现报数问题*/
  int count,num;
  LinkList pre, p;
  num=0; count=1;
  p=head;
  printf("\noutput data : ");
  while (num<n)
   { do                   /*累计报数*/
       {  count++;pre=p;    p=p->next; }while (count<m);
     pre->next=p->next;  /*报 m 的人出列*/
     printf("%3d",p->No);
     free(p);
     p=pre->next;
     count=1;
     num++;
   }
}
 main()
 { LinkList head;
   head=create();
   search (head);
   getchar();
   getchar();
}
```

【测试数据】

input n (总人数) : *8*

input m : *3*

output data : 3　6　1　5　2　8　4　7

2. 一元多项式的求导运算

【案例描述】

已知一个一元多项式，试设计一个程序对该多项式求导。

【案例要求】

（1）输入一元多项式，建立与其对应的有序链表。

（2）对多项式进行求导，建立并输出求导后的多项式。

【数据结构】

```
typedef struct pnode{
    float coef ;          /*系数*/
    int expn;             /*指数*/
```

```
    struct pnode *next;
}pnode,*polylink;
```

【案例分析】

可以选择带头结点的单链表或循环单链表存储多项式。头结点的指数值为-1，系数项存放多项式的项数。导数的运算原则如下：常数的导数为零，x^n 的导数等于 $n*x^{n-1}$。假设 qa 指向多项式的当前操作结点，则结点中各数据项的变化为：

（1）若 qa 所指结点的指数项为 0，则求导以后应从多项式中删除该结点，释放 qa。

（2）若 qa 所指结点的指数项不为 0，则求导以后，该结点的系数为该结点的原系数与该结点的原指数的乘积，该结点的指数减 1。

【案例实现】

```
#include  "stdio.h"
typedef struct pnode{
float coef ;        /*系数*/
int expn;           /*指数*/
struct pnode *next;
}pnode,*polylink;

void creatpolyn(polylink p,int m)
{ /*输入 m 项的系数和指数,建立一元多项式的有序单链表*/
  int i;  float coef;  int expn;
  polylink  tail, new;
  p->coef=m;  p->expn=-1;  tail=p;
  for(i=1;i<=m;i++) {
     new=( polylink)malloc(sizeof(pnode));
     printf("\n input coef(系数) and expn(指数) : ");
     scanf("%f%d",&coef, &expn);
     new->coef=coef;
     new->expn=expn;
     new->next=NULL;
     tail->next=new;
     tail=new;
  }
}

void derivative(polylink  p)
{  /*对一元多项式的有序链表 p 求导*/
 polylink pa,qa;
 pa=p->next;
 qa=p;
 while(pa)
 { if(pa->expn==0)  /*从多项式中删除*pa 结点并释放存储空间*/
       {   qa->next=pa->next;
           free(pa); p->coef--;
           pa=qa->next;
       }
   else            /*修改*pa 结点的系数和指数*/
      {    pa->coef*=pa->expn;
           pa->expn-=1;
           qa=pa;pa=pa->next;
```

```
      }
   }
}
void printpolyn(polylink p)
{    /*打印输出一元多项式 p*/
     int n=0;
     polylink q=p->next;
     while(q)
     {n++;
      if (n==1)
           { if (q->expn==0) printf("%.2f", q->coef);
             else
          if (q->expn==1)
             {  if (q->coef==1)  printf("x");
               else if (q->coef==-1)  printf("-x");
                       else { printf("%.2f", q->coef); printf("x"); }
             }
              else if (q->coef==1) printf("x^%d", q->expn);
              else if( q->coef==-1)  printf("-x^%d", q->expn);
              else  printf("%.2fx^%d", q->coef, q->expn);
            }/*if(n==1)*/
         else if (q->expn==0) printf("+%.2f", q->coef);
         else
         if (q->expn==1)
          { if (q->coef==1)    printf("+x");
            else if (q->coef==-1)  printf("-x");
                   else if (q->coef>0)  printf("+%.2fx", q->coef);
                        else printf("%.2fx", q->coef);
          }
            else if (q->coef==1)   printf("+x^%d", q->expn);
            else if( q->coef==-1)  printf("-x^%d", q->expn);
            else if (q->coef>0) printf("+%.2fx^%d", q->coef, q->expn);
            else   printf("%.2fx^%d", q->coef, q->expn);
     q =q->next;
   }/*while*/
}/*printpolyn*/
main()
{
polylink p;
int m;
printf("\ninput m (多项式的项数) : ");
scanf("%d",&m);
p=(polylink)malloc(sizeof(pnode));
creatpolyn(p,m);
printf("\np(x)=");
printpolyn(p);
derivative(p);
printf("\np'(x)=");
printpolyn(p);
}
```

【测试数据】

input m (多项式的项数) : *3*

input coef (系数) and expn(指数) : *4　3*

input coef (系数) and expn(指数) : *3　2*

input coef (系数) and expn(指数) : *2　1*

p(x)=4.00x^3+3.00x^2+2.00x

p'(x)=12.00x^2+6.00x+2.00

3. 集合的基本运算

【案例描述】

假设以两个递增有序排列的线性表 A 和 B 分别表示两个集合（即同一表中的元素值各不相同），现要求另辟空间构成线性表 C、D 和 E，其元素分别为 A 和 B 中元素的交集、并集和差集，且表 C、D 和 E 中的元素也递增有序排列。试对顺序表编写求 C、D 和 E 的算法。

【案例要求】

所设计的程序应实现如下基本功能：

（1）建立并输出交集 C。

（2）建立并输出并集 D。

（3）建立并输出差集 E。

【案例分析】

求交集 C 时，执行过程如下：

（1）i 标记顺序表 A 的当前元素，j 标记顺序表 B 的当前元素，k 计数顺序表 C 的元素个数；

（2）当顺序表 A 和 B 均未结束时，做如下处理：

①若 A 的元素小于 B 的元素，i++；

②若 A 的元素大于 B 的元素，j++；

③若 A 的元素等于 B 的元素，添加到 C 中。

求并集 D 时，执行过程如下：

（1）i 标记顺序表 A 的当前元素，j 标记顺序表 B 的当前元素，k 计数顺序表 D 的元素个数。

（2）当顺序表 A 和 B 均未结束时，做如下处理：

①若 A 的元素小于等于 B 的元素，将 A 的元素添加到 D 中；

②若 A 的元素大于 B 的元素，将 B 的元素添加到 D 中。

（3）将顺序表 A 或顺序表 B 的剩余元素，全部添加到 D 中。

求差集 E 时，执行过程如下：

（1）i 标记顺序表 A 的当前元素，j 标记顺序表 B 的当前元素，k 计数顺序表 E 的元素个数。

（2）当顺序表 A 未结束时，做如下处理：

①若 A 的元素不在 B 中，将 A 的元素添加到 E 中；

②若 A 的元素在 B 中，跳出本次循环。

【案例实现】

```
#include "stdio.h"
#define  MAXSIZE  100            /*顺序存储空间所能达到的最多元素数 */
typedef  int  ElemType;
typedef  struct{
   ElemType   elem[MAXSIZE];    /*存储空间*/
   int   length;                /*顺序表长度*/
} SqList;
void SqList_C (SqList A,SqList B,SqList *C)    /*求交集C*/
 {int i,j,k;
  i=1;j=1;k=0;
  while(i<=A.length&&j<=B.length)
  { if(A.elem[i]<B.elem[j]) i++;
    if(A.elem[i]>B.elem[j]) j++;
    if(A.elem[i]==B.elem[j])
    {/*当发现一个在A和B中都存在的元素,添加到C中*/
      (*C).elem[++k]=A.elem[i];
      i++;j++;
    }
  }/*while*/
(*C).length=k;
}/*SqList_C*/
void SqList_D(SqList A,SqList B,SqList *D)     /*求并集D*/
{ int i,j,k;
  i=1;j=1;k=0;
  while(i<=A.length&&j<=B.length)
  { if(A.elem[i]<=B.elem[j])
       { (*D).elem[++k]=A.elem[i]; i++; }
    else
       { (*D).elem[++k]=B.elem[j]; j++; }
  }/*while*/
 while(i<=A.length) {(*D).elem[++k]=A.elem[i]; i++; }
 while(j<=B.length) {(*D).elem[++k]=B.elem[j]; j++; }
(*D).length=k;
}/*SqList_D*/
void SqList_E(SqList A,SqList B,SqList *E)     /*求差集E*/
 {
  int i,j,k;
  i=1; k=0;
  while(i<=A.length)
  { j=1;
    while(j<=B.length && (A.elem[i]!=B.elem[j])) j++;
    if(j>B.length)
       (*E).elem[++k]=A.elem[i];
    i++;
  }/*while*/
(*E).length=k;
}/*SqList_E*/
void creat(SqList *L)                          /*生成一个集合*/
{
int i,m;
printf("\ninput m (集合元素数): ");
```

```
scanf("%d",&m);
printf("\ninput data : ");
for(i=1;i<=m;i++)
   scanf("%d",&(*L).elem[i]);
(*L).length=m;
}
void print(SqList L)                          /*输出一个集合中的元素*/
{
int i;
printf("(");
for(i=1;i<L.length;i++)
   printf("%d,",L.elem[i]);
printf("%d",L.elem[i]);                       /*输出集合的最后一个元素*/
printf(")");
}
main()
{
{SqList A, B, C, D, E;
creat(&A);
creat(&B);
SqList_C (A,B,&C);
printf("\nA∩B=");
print(C);
SqList_D (A,B,&D);
printf("\nA∪B=");
print(D);
SqList_E (A,B,&E);
printf("\nA-B=");
print(E);
getchar();
getchar();
}
```

【测试数据】

input m (集合元素数) : *4*

input data (集合元素): *3*　*5*　*8*　*11*

input m (集合元素数) : *7*

input data (集合元素): *2*　*6*　*8*　*9*　*11*　*15*　*20*

A∩B=(8, 11)

A∪B=(2, 3, 5, 6, 8, 8, 9, 11, 11, 15, 20)

A-B=(3, 5)

1.3 实验项目

1. 链表的维护与文件形式的保存

以链表结构的有序表表示某商厦家电部的库存模型。当有提货或进货的业务要求时，需要对该有序表及时进行维护。每个工作日结束之后，将链表结构的有序表中的数据以文件的形式保存；每天营业之初需要将文件形式的数据恢复成链表结构的有序表。

2. 自动预订飞机票的系统

设民航公司有一个自动预订飞机票的系统，该系统中有一个用双向链表表示的乘客表，表中结点按乘客姓氏的字母顺序相连。试为该系统设计一个程序，当任一乘客要订票时，修改乘客表。

实验 2　栈 和 队 列

2.1　实验基础知识

1. 顺序栈和链栈的特点及基本操作

栈是限定仅在表尾进行插入或删除操作的线性表。因此，对栈来说，表尾端有其特殊的含义，称为栈顶，相应地，表头端称为栈底。不含元素的空表称为空栈。

假设栈 $S=(a_1, a_2, \ldots, a_n)$，则称 a_1 为栈底元素，a_n 为栈顶元素。栈中元素按 $a_1, a_2, \ldots, a_n$ 的次序进栈，退栈的第一个元素应为栈顶元素。换句话说，栈的修改是按后进先出的原则进行的。因此，栈又称为后进先出的线性表。

利用一组地址连续地存储单元依次存放自栈底到栈顶的数据元素，同时附设指针 top 指示栈顶元素在顺序栈中的位置。

栈的链式存储结构称为链栈。链栈是不带头结点的单链表。

2. 队列的特点及基本操作

队列是一种先进先出的线性表，它只允许在表的一端进行插入，而在另一端删除元素。最早进入队列的元素最早离开。在队列中，允许插入的一端称为队尾，允许删除的一端则称为队头。

假设队列为 $Q=(a_1, a_2, \ldots, a_n)$，那么，$a_1$ 就是队头元素，a_n 则是队尾元素。队列中的元素是按照 $a_1, a_2, \ldots, a_n$ 的顺序进入的，退出队列也只能按照这个次序依次退出，也就是说，只有在 $a_1, a_2, \ldots, a_{n-1}$ 都离开队列之后，a_n 才能退出队列。

用链表表示的队列称为链队列，一个链队列需要两个分别指示队头和队尾的指针才能惟一确定。为了操作方便，给链队列添加一个头结点，并令队头指针指向头结点。因此，空的链队列的判决条件为队头指针和队尾指针均指向头结点。链队列的操作即为单链表的插入和删除操作的特殊情况，只是尚需修改队尾指针或队头指针。

在队列的顺序存储结构中，除了用一组地址连续的存储单元依次存放从队头到队尾的元素之外，还需附设两个指针 front 和 rear 分别指示队头元素和队尾元素的位置。初始化建空队列时，令 front=rear=0，每当插入新的队列尾元素时，尾指针增 1；每当删除队列头元素时，头指针增 1。因此，在非空队列中，头指针始终指向队头元素，而尾指针始终指向队尾元素的下一个位置。

2.2　实验案例及其分析

1. 括号匹配问题

【案例描述】

假设一个算术表达式中可以包含 3 种括号：圆括号“（”和“）”、方括号“［”和“］”

以及花括号“｛”和“｝”，且这3种括号可按任意的次序嵌套使用（如：…[…｛…｝…[…]…]…[…]…（…）…）。设计一个程序，判别给定表达式中所含括号是否配对。

【案例要求】

（1）将算术表达式保存在带头结点的单链表或数组中。

（2）通过顺序栈实现括号匹配问题的求解。

【案例分析】

判断表达式中括号是否匹配，可通过栈来实现，简单说是当读入的字符为左括号时进栈，为右括号时退栈。退栈时，若栈顶元素是其对应的左括号，则栈顶元素出栈；若不是则说明括号不匹配，算法结束。如此下去，当输入表达式结束时，若栈为空，则说明表达式括号匹配，否则说明表达式括号不匹配。

另外，由于本题是对表达式的括号匹配问题进行检查，所以对于输入的表达式中不是括号的字符一律不进行处理。

【案例实现】

```
#include  "stdio.h"
typedef struct node{
char ch;
struct node  *next;
}Lnode,*LinkList;              /*类型描述*/
LinkList  create(void)         /*生成单链表并返回*/
{char  ch1;
LinkList  head, tail, new;
head=(LinkList)malloc(sizeof(Lnode));
head->next=NULL;tail=head;
printf("\n请输入表达式并且以'#'作为结束符:\n");
scanf("%c",&ch1);
while(ch1!='#')
  {new=(LinkList)malloc(sizeof(Lnode));
   new->ch=ch1;
   tail->next=new;
   tail=new;
   scanf("%c",&ch1);
  }
 tail->next=NULL;
 return  head;
}
int Match(LinkList la){
/*表达式存在以 la 为头结点的单链表中,本算法判断括号是否配对*/
char s[30];                    /*s 为字符栈,容量足够大*/
int top=0;
LinkList  p;
p=la->next;                    /*p 为工作指针,指向待处理结点*/
while(p!=NULL)                 /*单链表未结束*/
 {switch (p->ch)
  {case '(':s[++top]=p->ch; break;
   case ')':if(top==0||s[top]!= '(')
                  {printf("\n(no)括号不配对\n"); return(0);}
```