全国高职高专规划教材

数据结构实训教程

Data Structures in Practice

孙巧萍 主 编
王爱冬 刘鲁楣 副主编

全国高职高专规划教材

科学出版社
www.sciencep.com

全国高职高专规划教材

数据结构实训教程

孙巧萍　主　编

王爱冬　刘鲁楣　副主编

科学出版社

北　京

内 容 简 介

本书是数据结构实验课教材，为“数据结构”课程上机实践提供理论与操作指导，可与采用C语言进行算法描述的各种版本的“数据结构”教材配套使用。

全书共分7章，前5章分别讨论线性表、栈和队列、串和数组、树、图等内容，第6章和第7章讨论各种查找和排序方法的算法实现与应用。本书内容由浅入深，采取循序渐进的方式培养学生的实践技能。

本书不仅可作为高职、高专计算机专业的配套教材，也是对本、专科相关专业学生，自考学员和专业教师颇有帮助的辅助教材。

图书在版编目(CIP)数据

数据结构实训教程/孙巧萍主编；王爱冬，刘鲁楣副主编. —北京：科学出版社，2003

（全国高职高专规划教材）

ISBN 7-03-012006-X

Ⅰ.数… Ⅱ.①孙…②王…③刘… Ⅲ.数据结构—高等学校：技术学校—教材 Ⅳ.TP311.12

中国版本图书馆CIP数据核字（2003）第066583号

策划编辑：李振格/责任编辑：丁 波

责任印制：吕春珉/封面设计：东方人华平面设计部

科学出版社 出版

北京东黄城根北街16号

邮政编码：100717

http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2003年8月第 一 版 开本：787×1092 1/16

2003年8月第一次印刷 印张：10 3/4

印数：1—5 000 字数：239 000

定价：15.00元

（如有印装质量问题，我社负责调换〈环伟〉）

全国高职高专规划教材编委会名单

主　任　俞瑞钊

副主任　陈庆章　蒋联海　周必水　刘加海

委　员　（以姓氏笔画为序）

王　雷　王筱慧　方　程　方锦明　卢菊洪　代绍庆

吕何新　朱　炜　刘向荣　江爱民　江锦祥　孙光弟

李天真　李永平　李良财　李明钧　李益明　余根墀

汪志达　沈凤池　沈安衢　张　元　张学辉　张锦祥

张德发　陈月波　陈晓燕　邵应珍　范剑波　欧阳江林

周国民　周建阳　赵小明　胡海影　秦学礼　徐文杰

凌　彦　曹哲新　戚海燕　龚祥国　章剑林　蒋黎红

董方武　鲁俊生　谢　川　谢晓飞　楼　丰　楼程伟

鞠洪尧

秘书长　熊盛新

本书编写人员名单

主　编　孙巧萍

副主编　王爱冬　刘鲁楣

此为试读，需要完整PDF请访问：www.ertongbook.com

前 言

“数据结构”作为一门课程，在20世纪70年代末进入大学课堂。它不仅是计算机专业的核心课程，也是其他理工科专业的热门选修课。作者在长期讲授“数据结构”这门课程的过程中深切地体会到，在整个教学活动中，学生解题能力和上机操作能力的培养是两个至关重要的环节，后者显得更为重要。学生仅仅学好本课程的理论知识是远远不够的，必须加强实践环节，从实验的成功和失败中获得锻炼，以提高复杂程序设计的技能及解决实际应用问题的能力。但从现状来看，目前国内“数据结构”实验课程教材比较缺乏，适合高职高专的此类教材更是寥寥无几，难以跟上教学实践的需求。没有合适的实训教程的指导，学生在做实验时往往会感到无从下手，达不到理想的实验效果，为此，作者结合自己多年来的教学经验编写了这本《数据结构实训教程》。本书力求在加强实验课教学环节上能有所突破，确保从抓实践环节上入手，注重培养学生的实际应用技能和综合解决问题的能力，使学生能熟练掌握和运用理论知识解决实际问题，达到学以致用的目的，能真正地为培养新世纪的适用型人才出一份力。

本书紧扣数据结构的理论知识，每类数据结构相关内容各成一章，结构清晰，内容按由易到难、逐层深入的方式安排，以激发读者探讨问题的兴趣。每章内容除包括实训知识准备、实训案例分析和实训项目三部分外，还单独列出习题，供学生课后复习巩固使用。为体现实训特色、突出实训重点，尤其是培养学生应用理论知识解决实际问题的能力，在每章的实训案例分析中，都给出了若干个实际问题的算法要点、算法及程序，从而让学生熟悉综合应用所学理论知识解决实际问题的方法和一般过程，达到锻炼综合应用能力的目的。

本书的宗旨是真正地教会学生如何做数据结构实验，怎样做好数据结构实验，书中所有算法及程序均采用C语言编写，可与各种版本用C语言描述的“数据结构”教材配套使用。

本实训教程由孙巧萍老师统编全稿，其中前3章由王爱冬老师执笔，第4、5章由孙巧萍老师执笔，第6、7章由刘鲁楣老师执笔。本书中的所有程序都在Visual C++ 6.0中调试通过。由于编者水平有限、时间仓促，书中难免存在一些不足之处，殷切希望广大读者批评指正。

编 者

2003年6月

目　录

第 1 章　线性表

线性表是最简单、最常用的基本数据结构，在设计有效算法、解决实际问题中有广泛的应用。理解线性表的逻辑结构、存储结构并熟练掌握线性表的基本操作是应用线性表解决实际问题的前提，本章的实践活动以巩固对线性表逻辑结构的理解，强化训练线性表的操作，培养实际应用的能力为宗旨，并为后续章节的实践打下基础。

实训概要

本章实训的目的在于验证线性表的两种存储结构及线性表的基本操作在两种存储结构上的实现，掌握以线性表作为数据结构解决实际问题的方法。

实训总体要求

- 实训中对于线性表的两种存储结构及其基本操作要进行验证
- 运行实训案例中的程序，理解实训案例中算法要点
- 认真分析实训项目中的内容，将其中的算法付诸实现，掌握线性表的实际应用
- 自主完成实训思考题

实训重点

- 线性表基本操作的实现
- 线性表的实际应用

1.1　实训知识准备

简单地讲，线性表是 n 个元素的有限序列。其特点是在数据元素的非空集合中：

① 存在惟一的一个称为“第一个”的数据元素。

② 存在惟一的一个称为“最后一个”的数据元素。

③ 除第一个元素外，集合中的每个元素均只有一个直接前趋。

④ 最后一个元素外，集合中的每个元素均只有一个直接后继。

其中的后两点正是线性表逻辑结构的体现。线性表逻辑结构的存储实现分为顺序（顺序表）和链式（链表）两种。

1.1.1　顺序表

1. 基本知识

顺序表是线性表的顺序存储结构，是指用一组连续的存储单元依次存储线性表的数

此为试读，需要完整PDF请访问：www.ertongbook.com

据元素。在顺序表中，逻辑关系相邻的两个元素在物理位置上也相邻，元素之间的逻辑关系是通过下标反映出来的。在这种存储方式下，线性表逻辑关系的表达式不占用另外的存储空间。一般地，线性表的第 i 个元素 a_i 的存储位置为

$$LOC(a_i)=LOC(a_1)+(i-1)*l$$

其中，$LOC(a_1)$ 是线性表的起始位置，l 为每个元素所占空间的大小，因此，对线性表中的任一个元素都可以随机存取。如图 1.1 所示。

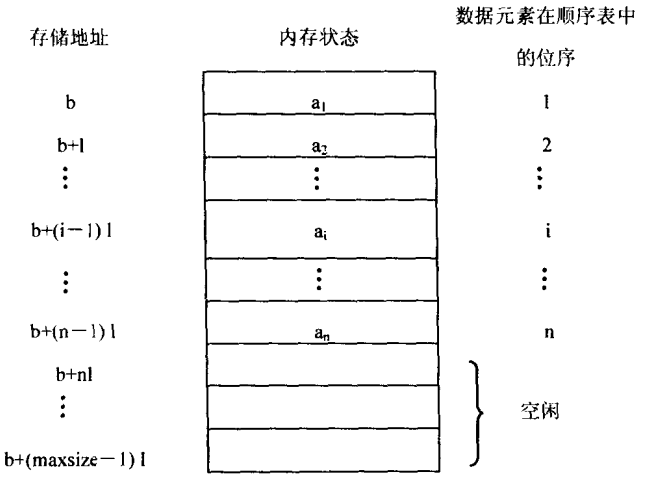


图 1.1 线性表的顺序存储结构示意图

由于高级程序设计语言中的数组类型也有随机存取的特性，因此，通常都用数组来描述数据结构中的顺序存储结构。

数据类型定义如下。

```
typedef int datatype
#define maxsize 线性表可能的最大长度
typedef struct
{
   datatype data[maxsize];
   int last;
} sqlist;
```

2. 基本操作

(1) 顺序表插入

在顺序存储结构的线性表 L 中第 i 个位置上插入 x。

$$(a_1,a_2,\ldots,a_{i-1},a_i,\ldots,a_n) \rightarrow (a_1,a_2,\ldots,a_{i-1},x,a_i,\ldots,a_n)$$

```
int ins_sqlist (sqlist *L, datatype x,int i)
{
   int j;
   if(L->last>=maxsize-1)
   {
      printf("溢出");
      return NULL;                          //溢出
```

```
    }
    else
    if((i<1)||(i> L-> last+1))
    {
        printf("出错");
        return NULL;                          //非法位置
    }
    else
    {
        for(j=L->last;j>=i-1;j--)
            L->data[j+1]= L->data[j];         //结点后移
        L->data[i-1]=x;                       //插入
        L->last=L->last+1;                    //长度加1
    }
    return(1);
}
```

（2）顺序表删除

在顺序存储结构的线性表 L 中删除第 i 个数据元素。

$(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n) \rightarrow (a_1,a_2,\cdots,a_{i-1},a_{i+1},\cdots,a_n)$

```
int del_sqlist(sqlist * L,int i)              //顺序表删除
{
    int j;
    if( (i<1) || (i>L->last+1) )
    {
        printf("出错");
        return NULL;
    }
    else
    {
        for(j=i;j<= L->last;j++)
          L->data[j-1]:= L->data[j];
        L->last - -;
    }
    return(1);
}
```

（3）顺序表查找

在顺序存储结构的线性表 L 中查找第 1 个值为 x 的数据元素。

```
int loc_sqlist(sqlist *L,datatype x)          //查找第1个值为x的数据元素
{
   int i;
   i=1;
   while( (i<=L->last) && (L->data[i]!=x))
     i=i+1;
   if(i<=L->last)  return(i);                 //找到
   else  return(0);                           //未找到
}
```

1.1.2 链表

链表是线性表的链式存储结构，其特点是用一组任意的存储单元（可以是连续的，也可以是不连续的）存储线性表中的数据元素。链表有带头结点和不带头结点、循环和非循环、单向和双向之分。

1. 单链表

（1）基本知识

线性表中元素在单链表中的存储映像称为结点，每个结点包括两个域：数据域和指针域，其中数据域存储数据元素的信息，指针域存储该结点的直接后继结点的起始位置。

线性表（a_1，a_2，…，a_n）所对应的带头结点的单链表，如图 1.2 所示。

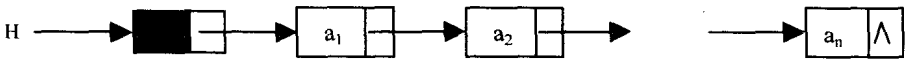
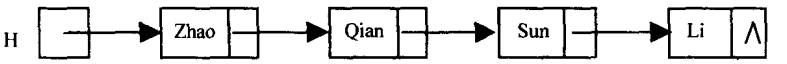


图 1.2 带头结点的单链表

图 1.3 给出了一个不带头结点的单链表实例的逻辑状态及其在内存中的存储表示。



（a）单链表实例

头指针 H 21

存储地址	数据域	指针域
5	Qian	35
21	Zhao	5
1	Li	NULL
35	Sun	1

（b）单链表的存储示例

图 1.3 线性链表的存储示例

数据类型说明如下。

```
typedef char datatype
typedef struct node
{
  datatype data;
     struct node *next;
} linklist;
```

（2）基本操作

① 单链表的建立。

```
linklist  *creat_list()                    //建立单链表
{
  char ch;
  linklist *head, *s, *r;
```

```
  head=NULL;                                          //链表开始为空
  r=NULL;
  while(1)
  {
     ch=getchar();                                    //读入结点的值
     if(ch=='$')                                      //'$'为输入的结束标志
        break;
     s= (linklist *) malloc(sizeof(linklist));  //生成新结点,由s所指向
     s->data=ch;
     s->next=NULL;
     if(head==NULL)   head=s;                         //把s结点链到单链表中
     else  r->next=s;
     r=s;                                             //尾指针指向表尾
  }
  if(r!=NULL)   r->next=NULL;                         //尾结点指针置空
  return head;
}
```

② 单链表的插入。

在单链表中第 i（i≥1）个结点之前插入一个值为 x 的结点。

```
linklist  *ins_linklist1(linklist *head, int i, datatype x)
{
   linklist *s, *p;
   int j;
   s=(linklist *)malloc(sizeof(linklist));  //生成新结点
   s->data=x;
   s->next=NULL;
   if (i==1)
   {
      s->next=head;                                   //插入到表头前
      head=s;
   }
   else
   {
     j=1;
     p=head;                                          //指针初始化，j为计数器
     while ((p!=NULL) && (j<i-1))
     { j=j+1;
       p=p->next;
     }                                                //查找第i-1个结点，由p指向
     if(p!=NULL)
     { s->next=p->next;
       p->next=s;
     }                                                //若查找成功,则完成插入
     else  printf("出错")                             //没有找到，出错
   }
     return head;
}
```

在值为 a 的结点前插入值为 x 的结点。

```
linklist  *insert_linklist2(linklist  *head, datatype a,datatype x)
{
   linklist *p,*s;
   s=(linklist *)malloc(sizeof(linklist));  //建立新结点s
   s->data=x;
   s->next=NULL;
   p=head;
   if(head==NULL)
     head=s;                                //插入到空表
   else
   if(head->data==a)                        //插入到表头
   {
      s->next=head;
      head=s;
   }
   else          //查找结点a , p指向结点a, q指向p所指结点的前趋结点
   {
      while ((p->data!=a) && (p->next!=NULL))
      {
        q=p;
        p=p->next;
      }
      if (p->data==a)
      {
         s->next=p;
         q->next=s;
      }                                     //插入到表的中间
      else  p->next=s;                      //插入到表尾
      }
      return head;
  }
```

③ 单链表的删除。

删除单链表中值为 x 的结点。

```
linklist  *del_linklist1(linklist *head,datatype x)
{
  linklist  *p, *q;
  if(head==NULL)
  printf("'链表下溢!");                     //当链表为空时，作下溢处理
  if(head->data==x)
  {
     p=head;
     head=head->next;                       //删除表头结点
     free(p);
  }
  else
  {
    q=head;
    p=head->next;                           //查找值为x的结点
    while((p!=NULL) && (p->data!=x))
```

```
    {
      q=p;
      p=p->next;
    }                                          //找到时,p指向该节点,q指向前一结点
    if(p!=NULL)
    {
       q->next=p->next;
       free(p);
    }
    else printf("未找到!");
  }
  return head;
}
```

删除单链表中第i（1≤i≤表长）个结点。

```
linklist *del_linklist2(linklist *head, int i) //删除单链表中第i (1≤i≤表长)个结点
{
  linklist *p, *q;
  int  j;
  p=head;
  j=1;                          //指针初始化，j为计数器
  while ((p->next!=NULL) && (j<i))
  {
    p=p->next;
    j=j+1;
  }                             //寻找第i个结点，并令p指向它的前驱结点
  if((p->next==NULL) || (j>i))
    printf("出错");
  else
  {
    q=p->next;
    p->next=p->next->next;
    free(q);                    //释放被删结点
  }
  return head;
}
```

④ 单链表的查找。

在表头指针为head的单链表中查找值为x的元素。

```
linklist *locate(linklist *head, datatype x)
{
   linklist *p;
   p=head;
   while(p!=NULL && p->data!=x)
      p=p->next;
  return p;
}
```

此为试读,需要完整PDF请访问: www.ertongbook.com