高等院校实验课教材

数据结构（C）实验教程

李业丽　郑良斌　主编

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

数据结构(C)实验教程

李业丽　郑良斌　主编

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

图书在版编目(CIP)数据

数据结构(C)实验教程 / 李业丽,郑良斌主编. —北京:北京理工大学出版社,2005.12

高等院校实验课教材

ISBN 7-5640-0487-8

Ⅰ.数… Ⅱ.①李… ②郑… Ⅲ.①数据结构-高等学校-教材②C语言-程序设计-高等学校-教材 Ⅳ.①T11.12②TP312

中国版本图书馆 CIP 数据核字(2005)第 135883 号

出版发行 / 北京理工大学出版社
社　　址 / 北京市海淀区中关村南大街 5 号
邮　　编 / 100081
电　　话 / (010)68914775(办公室)　68944990(批销中心)　68911084(读者服务部)
网　　址 / http://www.bitpress.com.cn
电子邮箱 / chiefeditor@bitpress.com.cn
经　　销 / 全国各地新华书店
印　　刷 / 北京圣瑞伦印刷厂
开　　本 / 787 毫米×1092 毫米　1/16
印　　张 / 13.75
字　　数 / 325 千字
版　　次 / 2005 年 12 月第 1 版　2005 年 12 月第 1 次印刷
印　　数 / 1～4500 册　　责任校对 / 郑兴玉
定　　价 / 22.00 元　　责任印制 / 吴皓云

前　言

《数据结构》是计算机及相关专业的一门重要专业基础课程，也是一门必修的核心课程。在计算机科学的各领域中，都将会用到各种不同的数据结构，学好数据结构这门课程，对从事计算机技术及相关领域的工作人员来说非常重要。

由于数据结构的原理和算法比较抽象，理解和掌握其中的原理就显得较为困难。学习这门课程，实验是非常关键的环节，上机实验是理解算法最佳的途径之一。为了帮助读者更好地学习本课程，理解和掌握算法设计所需的技术，作者通过多年的教学实践，收集、整理进而编写了这本《数据结构(C)实验教程》，希望读者通过上机实验加强对数据结构算法的理解，提高读者分析问题和解决问题的能力。

本书根据数据结构课程教学内容，总结出每章的内容要点，有针对性地设计了一些数据结构实验，对于每个实验，给出实验内容与要求、知识要点、实现提示、参考程序及思考与提高，所有的源程序都在 Turbo C 和 Visual C++ 6.0 环境下运行通过。通过这些实验，可以使读者了解并学会如何运用数据结构知识去解决现实世界中的一些实际问题，并具备设计较复杂算法的基本能力。在本书的附录中给出了参考实验报告模板，培养学生按照规范的形式书写实验报告的习惯。

本书的一个重要的特点是根据学生的基础知识、兴趣爱好，将实验分成基础实验和提高实验。基础实验主要验证数据结构课程中的基本算法，练习巩固课程内容。提高实验设计一些与实际问题紧密联系的难易程度不同的实验，读者可以根据自己的兴趣爱好与知识水平，自己选择实验题目。读者通过完成一系列的实验，巩固基础理论知识，培养分析、解决实际问题的能力，培养对复杂问题进行程序设计的能力。

本书既可以作为高等院校各类相关专业本科生、专科生学习数据结构的上机实验指导，也可以作为相关专业自学考试、研究生入学考试、计算机技术与软件专业技术资格(水平)考试、计算机等级考试(三级或四级)应试复习资料，同时也可供各类学习数据结构的人员参考使用。选用该教材上机实验的学校，可以根据学校自身的条件，在实验题目中有针对性地选一部分或全选。

本书是由北京印刷学院多位在第一线讲授数据结构课程的具有丰富教学经验的教师共同编写。本书共分 9 章。第 1 章介绍了线性表，第 2 章介绍了栈和队

列，第3章介绍了串、多维数组和广义表，第4章介绍了树与二叉树，第5章介绍了图，第六章介绍了查找，第7章介绍了排序。第1章由舒后编写，第2章由范士喜编写，第3章由杨树林编写，第4章由齐亚莉编写，第5章由刘华群编写，第6章由郑良斌编写，第7章由李业丽编写。全书由李业丽、郑良斌负责统稿、定稿。在本书的编写过程中，得到了北京印刷学院计算机系其他老师的大力支持，在此深表感谢。

由于作者水平有限、时间仓促，本书难免存在一些缺点和错误，恳请读者及同行批评指正。

编　　者

2005年8月于北京

目　录

第一章 线 性 表

1.1 内 容 要 点

线性表（Linear List）是最简单且最常用的一种数据结构。这种结构具有下列特点：存在一个唯一的没有前驱的（头）数据元素；存在一个唯一的没有后继的（尾）数据元素；此外，每一个数据元素均有一个直接前驱和一个直接后继数据元素。

1.1.1 线性表的定义及其运算

1. 线性表的定义

线性表是 n（n≥0）个数据元素 $a_1,a_2,\cdots,a_n$ 组成的有限序列。其中 n 称为数据元素的个数或线性表的长度，当 n=0 时称为空表，n>0 时称为非空表。通常将非空的线性表记为（$a_1,a_2,\cdots,a_n$），其中的数据元素 a_i（1≤i≤n）是一个抽象的符号，其具体含义在不同情况下是不同的，即它的数据类型可以根据具体情况而定。

2. 线性表的特征

从线性表的定义可以看出线性表的逻辑特征：

① 有且仅有一个开始结点(表头结点) a_1，它没有直接前驱，只有一个直接后继；

② 有且仅有一个终端结点(表尾结点) a_n，它没有直接后继，只有一个直接前驱；

③ 其他结点都有一个直接前驱和直接后继；

④ 元素之间为一对一的线性关系。

3. 线性表的运算

常见线性表的运算有：

① 置空表（初始化）setnull（&L） 将线性表 L 置成空表。

② 求长度 length（L） 求给定线性表 L 的长度。

③ 取元素 get（L，i） 若 1≤i≤length(L)，则取第 i 个位置上的元素，否则取得的元素为 NULL。

④ 定位函数 locate（L，X） 在线性表 L 中查找值为 X 的元素位置，若有多个值为 X，则以第一个为准，若没有，位置为 0。

⑤ 插入 insert（&L，X，i） 在线性表 L 中第 i 个数据元素之前插入一个值为 X 的新元素。

⑥ 删除 delete（&L，i） 删除线性表 L 中第 i 个元素。当 1≤i≤len 时，删除成功，返回被删元素的值，否则返回 NULL。

对线性表的操作还有很多，比如取前趋、取后继及排序等。

1.1.2 线性表的顺序存储结构

线性表的顺序存储结构，也称为顺序表。其存储方式为：在内存中开辟一片连续存储空间，但该连续存储空间的大小要大于或等于顺序表的长度，然后让线性表中第一个元素存放在连续存储空间第一个位置，第二个元素紧跟着第一个之后，其余依此类推。

可见，在线性表上，逻辑关系相邻的两个元素在物理位置上也相邻——线性表顺序存储的特点。

很容易确定每个数据元素在存储单元中与起始地址的相对位置：假设线性表中元素为（$a_1,a_2,\cdots,a_n$），设第一个元素 a_1 的内存地址为 $Loc(a_1)$，而每个元素在计算机内占 C 个存储单元，则第 i 个元素 a_i 的首地址为 $Loc(a_i)=Loc(a_1)+(i-1)\times C$　(其中 $1\leqslant i\leqslant n$)。

显然，顺序表便于进行随机访问，故线性表的顺序存储结构是一种随机存储的结构。

1. 顺序表的结构

下面用 C 语言，给出顺序表的定义：

```
#define MAXSIZE   maxlen
typedef   int elemtype;
typedef struct
{ elemtype vec[maxlen];
   int len;          //顺序表的长度
 }sequenlist;
```

有了线性表的顺序存储结构后，下面就可以讨论如何在这种结构上实现有关数据元素的运算问题。重点讨论线性表元素的插入和删除。

2. 顺序表的基本运算

（1）插入运算

线性表的插入是指在表的第 i 个位置上插入一值为 x 的新元素，通常在第 i 个元素 ($1\leqslant i\leqslant n+1$) 插入一个新元素时，需要有 n-i+1 个元素进行移动。通过以下步骤完成顺序表的插入运算：

① 将 $a_n\sim a_i$ 共 n-i+1 个元素依次向下移动，为新元素让出位置；

② 将 x 置入空出的第 i 个位置；

③ 修改 len 值（即修改表长），使之加 1。

算法描述：

```
  int   insert(sequenlist *L, int   i, elemtype x)
{ int j;
  if((*L).len>=maxlen)
    {printf("the list is overflow\n");
     return NULL;
     }
  else if((i<1)||(i>(*L).len+1))
                { printf("position is not corrent.\n");
                   return NULL;
```

```
                }
else{    for(j=(*L).len-1;j>=i-1;j--)
                    (*L).vec[j+1]=(*L).vec[j];          //元素后移
           (*L).vec[i-1]=x;                    //插入元素 x
           (*L). len++;                    //表长度增加 1
          return 1;
      }
 }
```

性能分析：

插入算法花费的时间，主要在于循环中元素的后移（其他语句花费的时间可以省去），即从插入位置到最后位置的所有元素都要后移一位，在空出的位置插入元素值 x。但是，插入的位置是不固定的，当插入位置 i= 1 时，全部元素都得移动，需 n 次移动，当 i=2 时，移动 n-1 个元素，依次类推，当 i=n 时，仅需移动元素一次。设 p_i 为在第 i 个数据元素之前插入一个元素的概率，假设在顺序表的任何位置上插入元素都是等概率的，即 $p_i=1/(n+1)$，则在长度为 n 的顺序表中插入一个元素时所需的平均移动次数为

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

可见在顺序表上做插入操作需移动表中一半的数据元素，显然时间复杂度为 O(n)。

（2）删除运算

线性表的删除运算是指将表中第 i 个元素从线性表中去掉，i 的取值范围为 $1\leqslant i\leqslant n$。通过以下步骤完成顺序表的删除运算：

① 将 $a_{i+1}\sim a_n$ 顺序向上移动；

② 修改 len 值（即修改表长），使之减 1。

算法描述：

```
 delete(sequenlist *L, int  i)
{int j;
if((*L).len==0)
  { printf("empty list\n");
   return NULL;
  }
else  if((i<1)||(i>(*L).len))
        { printf("delete fail\n");
          return NULL;
         }//删除的元素越界
else {
        for(int j=i;j<=len-1;j++)
              (*L).vec[j-1]=(*L).vec[j];                    //元素前移
```

```
        (*L).len--;                                    //表长度减 1
        return 1;
      }
  }
```

性能分析：

与插入运算相同，删除第 i 个元素时，后面的元素 a_{i+1}～a_n 都要向上移动一个位置，共移动了 n-i 个元素，故平均移动次数：

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

这说明在顺序表上作删除运算大约需要移动表中一半的元素，显然该算法的时间复杂度为 O(n)。

1.1.3 线性表的链式存储结构

线性表的链式存储结构，也称为链表。链式存储是用一些任意的存储单元来存储线性表中的元素，这些单元在物理上可以是连续的，也可以是不连续的。为了能正确地描述元素之间的逻辑关系，除了存储元素本身的信息外，还需存储指示元素之间逻辑关系的信息。这两部分信息组成数据元素 a_i 的存储映像，称为结点。

在链表中，每个结点所占的存储空间分为两部分：存放数值的域称为数据域，存放结点之间相互关系的域称为指针域。在定义的链表中，若只含有一个指针域来存放下一个元素地址，称这样的链表为单链表或线性链表。

1. 单链表结构

线性链表中的结点结构可描述为

Data	next

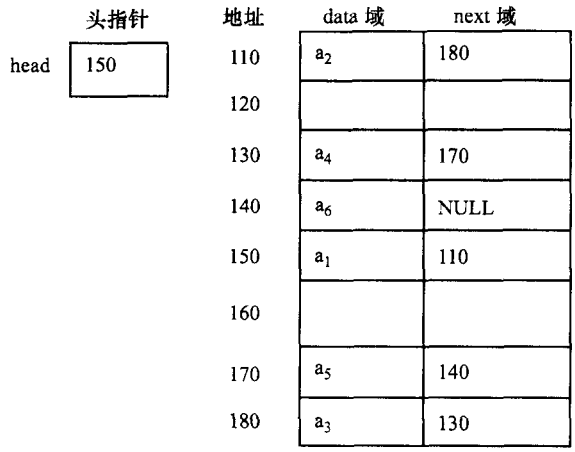
其中，data 域用来存放结点本身信息，类型由具体问题而定；next 域用来存放下一个元素地址，即存储指示其直接后继的信息。可见，用单链表表示线性表时，数据元素之间的逻辑关系是通过链结点中的指针反映出来的，即指针是数据元素之间逻辑关系的映像。图 1-1、图 1-2 给出单链表结构示意图。

2. 单链表上的基本运算

为了便于实现各种运算，通常在单链表的第一个结点前增设一个附加结点，称为头结点，它的结构与表结点相同，其数据域可不存储信息，也可存储如线性表长度一类的附加信息。单链表设置头结点的作用归纳如下：

① 头指针是指向头结点的非空指针，无论链表是否为空，头指针始终保持值不变，因此头指针的处理方法对空表和非空表的操作是一致的。这与不带头结点的单链表为空时头指针为空不同。

② 首结点的地址存放在头结点（可看作首结点的前驱结点）的指针域中，故对该结点的操作与对表中其他结点的操作一致，无须进行特殊处理。



· 图 1-1　单链表示意图

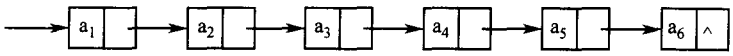


图 1-2　单链表的逻辑表示

用 C 语言描述结点结构如下：

```
typedef int elemtype;  //定义新类型，类型名为 elemtype
typedef struct node
 { elemtype data;    //数据域
   struct node *next; //指针域
  }linklist;
```

（1）单链表的构建

链表与顺序表不同，它是一种动态管理的存储结构，链表中的每个结点占用的存储空间不是预先分配，而是运行时系统根据需求而生成的，因此建立单链表从空表开始，每读入一个数据元素则申请一个结点，然后插在链表的尾部。

算法描述：

```
linklist   *creatlist(int n)
 {
   int x,k;      //设数据元素的类型为 int
  linklist   *head, *r, *p;
  p=(linklist *)malloc(sizeof(linklist));  //构建头结点空间
  head=p;
  p->next=NULL;
  r=p;
  for(k=1;k<=n;k++)          //循环构建 n 个结点
  { printf("input value:\n");
```

```
    scanf("%d",&x);
        p=(linklist *)malloc(sizeof(linklist));
      p->data=x;
      p->next=NULL;
      r->next=p;
      r=r->next;
     }
    return(head);
}
```

（2）单链表上元素的查找

从头指针开始，一般有两种查找方法：按元素的序号和按某个给定值。

按序号查找：

设单链表的长度为 n，要查找表中第 i 个结点，算法思想如下：

从头结点开始顺链扫描，用指针 p 指向当前扫描到的结点，用 j 作统计已扫描结点的计数器，当 p 扫描下一个结点时，j 自动加 1。

P 的初值指向头结点，j 的初值为 0。

当 j=i 时，指针 p 所指的结点就是第 i 个结点。

算法描述:

```
linklist    *find (linklist *head, int i)
{ int j;
  linklist *p;
  p=head->next;
  j=1;
  While((p!=NULL) && (j<i))
    { p=p->next;
      j++;
     }
  return p;
}
```

按值查找：

在链表中，查找是否有结点等于给定值 key 的结点，若有的话，则返回首次找到的值为 key 的结点的存储位置；否则返回 null。

算法思想

从开始结点出发，沿着链表将逐个结点的值和给定值 key 作比较。

算法描述：

```
    linklist *locate(linklist *head, elemtype x)
    { linklist *p;
        p=head->next;
```

```
while(p)      也可改为      while(p!=NULL && p->data!=x)
                                  p=p->next;
 { if(p->data!=x)
        p=p->next;
   else
        break;
 }
return p;
}
```

（3）单链表的插入运算

类似于查找运算，单链表的插入运算也可分两种方式来处理，一种按给定值插入，另一种按给定序号插入，根据给定的问题而定。下面主要讨论按给定值来处理。

① 已知线性链表 head，在 p 指针所指向的结点后插入一个元素 x。在一个结点后插入数据元素时，称为后插入法。操作过程如图 1-3 所示。

算法描述：

```
//将值为 x 的新结点插入到值为 k 的结点之后。
void insert( linklist *head, elemtype x, elemtype k)
 { linklist *p,*s;
   s=(linklist *)malloc(sizeof(linklist));  //生成新结点 x
   s->data=x;
  p=head->next;
if( p= =NULL)  //若为空表，则新结点作为表中唯一的结点
{ head->next=s;
   s->next=NULL;
   }
else { while(p && p->data!=k)
            p=p->next;
        if(p->data==k)
        { s->next=p->next;
          p->next=s;
        }
else //没有找到值为 k 的结点，将新结点插入到表尾
      {   p->next=r;
          r->next=NULL;
       }
    }
}
```

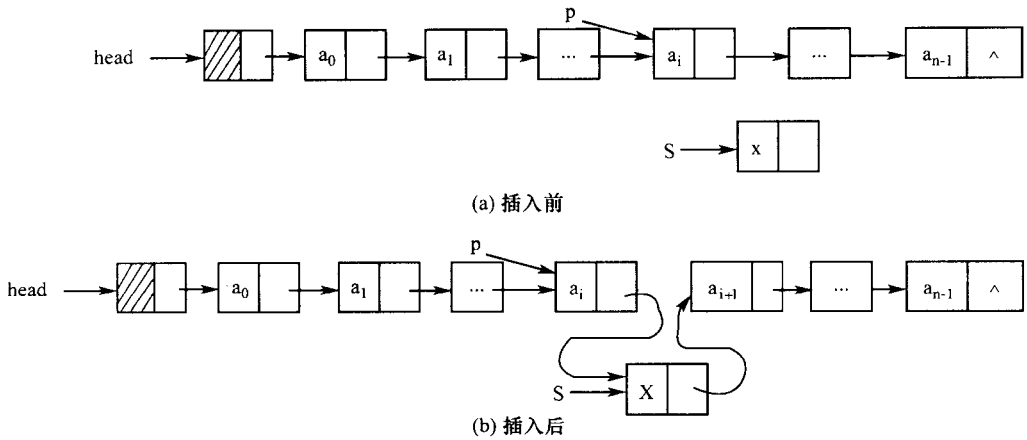


图 1-3 单链表的后插入

② 已知线性链表 head，在 p 指针所指向的结点前插入一个元素 x 称为前插，前插时必须从链表的头结点开始，找到 P 指针所指向的结点的前驱。设一指针 q 从附加头结点开始向后移动进行查找，直到 p 的前趋结点为止。然后在 q 指针所指的结点和 p 指针所指的结点之间插入结点 s。插入过程如图 1-4 所示。

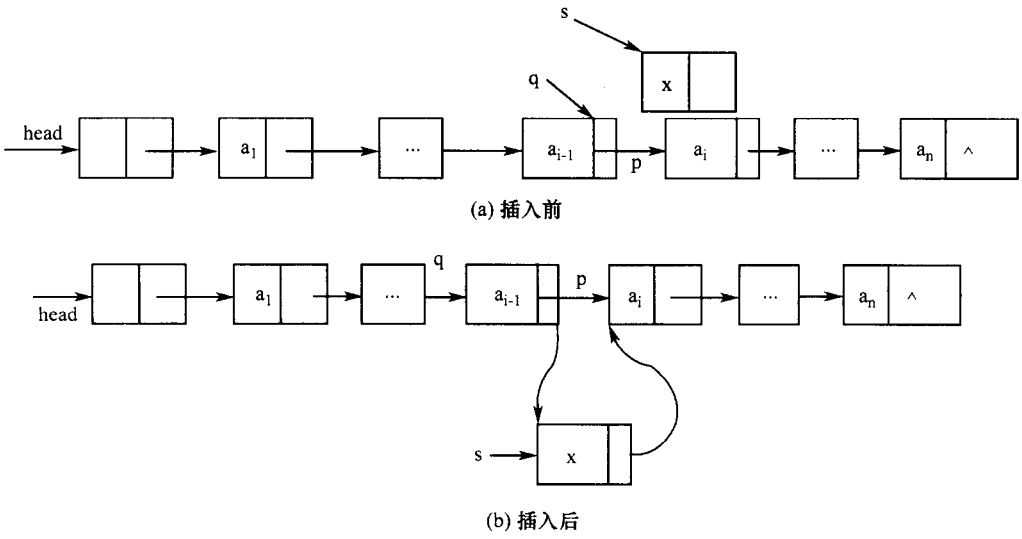


图 1-4 单链表的前插入

算法描述：

```
//将值为 x 的新结点插入到值为 k 的结点之前。
void insert( linklist *head, elemtype x, elemtype k)
{ linklist *q,*p,*s;
  s=(linklist *)malloc(sizeof(linklist));   //生成新结点 x
s->data=x;
if( head->next= =NULL)   //若为空表，则新结点作为链表中唯一的表结点
```