★大学计算机学习指导系列★

复习考研 自考

数据结构

学习与解题指南

殷新春 等 编著

华中科技大学出版社
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY PRESS
E-mail: hustpp@wuhan.cngb.com

大学计算机学习指导系列

数 据 结 构

学习与解题指南

殷新春　汤克明
杨晓秋　李　杰　编著

华中科技大学出版社

图书在版编目(CIP)数据

数据结构学习与解题指南/殷新春　等编著
武汉:华中科技大学出版社，　2001 年 3 月
ISBN 7-5609-1956-1

Ⅰ. 数…
Ⅱ. ①殷…　②汤…　③杨…　④李…
Ⅲ. 数据结构-高等学校-教学参考资料
Ⅳ. TP311.12

数据结构学习与解题指南　　殷新春　等编著

责任编辑:沈旭日　周芬娜　　封面设计:潘　群
责任校对:蔡晓瑚　　责任监印:张正林

出版发行:华中科技大学出版社
武昌喻家山　邮编:430074　电话:(027)87545012

录　　排:华中科技大学惠友科技文印中心
印　　刷:湖北省新华印刷厂

开本:787×960　1/16　　印张:22.75　　字数:374 000
版次:2001 年 3 月第 1 版　　印次:2001 年 3 月第 1 次印刷　　印数:1—5 000
ISBN 7-5609-1956-1/TP・333　　定价:26.80 元

内 容 提 要

本书系统地介绍了数据结构的基本概念、逻辑结构和存储结构，通过大量的题例分析阐述了数据结构解题的思想与方法，以及在算法设计中如何根据实际问题及相应的操作选择合理的数据结构。每章都附有自测练习及参考答案，最后一章附有模拟试卷及参考答案。

本书可作为计算机及相关专业数据结构课程的教学参考书，也可供参加自学考试、硕士研究生入学考试、等级考试的各类人员及计算机应用技术人员参考。

前 言

随着计算机科学和技术的发展，计算机的功能不断增强，运算速度不断提高，计算机的应用已经渗透到了社会的各个方面，成为当今信息社会的最显著的特征。

计算机加工处理的对象是数据，而数据具有一定的结构，数据的组织、存储和运算是设计和实现编译程序、操作系统、数据库系统及其它软件系统的重要基础。因此，介绍这些内容的数据结构是计算机及相关专业的最重要的专业基础课程，对计算机学科的学习起着承前启后的作用。

由于数据结构的原理和算法较抽象，且这门课程一般在低年级开设，而其前修课程中所涉及的专业知识又不多，因而加大了学习难度。数据结构课程教学中的一种常见现象是：理解授课内容并不困难，但一接触习题，往往不是无从下手，就是解答中出错很多。实际上，在理解课程内容与能够较好地完成习题之间存在着明显的差距，而算法题完成的质量与基本的程序设计的素质培养是密切相关的。

为了帮助读者更好地学习本课程，充分理解数据结构的原理，掌握算法设计所需的技术，为整个专业课程的学习打好基础，我们根据多年的教学经验编写了本书，以期能通过对概念的梳理、典型习题的解答，帮助读者深化对基本概念的理解，掌握求解数据结构问题的思路和方法，提高分析和解决问题的能力。

全书共分 12 章，第 1 章是预备知识，复习 Pascal 语言程序设计中的动态链表结构和递归编程技术，Turbo Pascal 中的单元技术，为学习数据结构打好坚实的基础；第 2 章是概述，讨论数据结构的基本概念及算法设计与分析的常见方法；第 3 章至第 8 章分别讨论线性表、栈和队列、串、数组和广义表、树和二叉树、图等几种基本的数据结构；第 9 章和第 10 章讨论查找和内排序；第 11 章讨论文件结构；第 12 章对数据结构做简要的复习，并提供了两份模拟试卷，供学完全书时自测。精选的例题以达到如下 3 个目的：①帮助理解数据结构课程的内容，强化基本概念；②训练程序设计技术，培

养良好的程序设计风格；③正确掌握对简单的应用问题合理选择数据结构的方法。每章的学习自评供读者在复习完本章的内容后自行测试对基本内容的掌握程度。

本书可作为计算机及相关专业学生学习数据结构课程的参考书，也是报考计算机专业硕士研究生的考生极具价值的复习书，同时也适合于数据结构课程自学考试者和计算机等级考试者研习。但是，本书不是教材，无法替代教科书，因此在使用本书时应注意：

1. 学习本课程之前，应对本课程的先修课程“Pascal 语言程序设计”进行系统的总结与复习；特别应读完本书第 1 章中的相关内容，从而对贯穿数据结构课程始终的链表结构和递归技术有深入的理解和熟练的掌握；同时，应注意类 Pascal 语言与标准 Pascal 语言之间的区别与联系。

2. 学习每一种数据结构时，首先应该掌握数据元素之间的逻辑关系是什么？在存储器中如何表示它们？在不同的存储结构上怎样实现对数据的运算？然后要掌握对不同的应用问题选择合理的数据结构以达到时、空性能的最佳。

3. 学习每一算法时，首先应理解问题的本身，然后自己构思一下解题的思路，最后再阅读算法。理解算法以后，不要急于向下看，而是停下来，想一想：该算法从逻辑上可分为几块？每块的功能是什么？是如何实现的？有没有其它方法？如果有新的想法应该记下来，与同学交流，与教师讨论。

4. 学习完教科书的一章后，再阅读本书相关的内容进行复习与自评。自评结果不理想时，千万别放弃，更不要急于继续向后看，应回过头来有目的有计划地重看教科书，重看学习导引，重做题目，直到本章内容掌握为止。

5. 学完本课程后，应对本课程的内容进行全面系统的小结和复习，重看本书，做完最后一章的模拟试卷。进行总复习时，要对比各种数据结构之间的异同及相互关系，以加深对各种数据结构的理解。

6. 在教学内容组织方面，各种教材都有不同的侧重面，甚至有些概念的表达也不尽相同（如一般树能否有空树？森林和一般树的遍历有几种？等等），读者应根据教科书或教师的讲授来学习、了解其中的异同。

最后应当强调的是，学习方法主要靠自己摸索。多总结，多思考，勤上机，勤交流，是把数据结构这门课程真正学好的关键。

由于作者水平有限，书中难免还存在一些缺点和错误，殷切希望广大读者批评指正。

编著者

2000 年 11 月

目 录

第 1 章

预备知识

本章基本内容 Pascal 语言程序设计中的动态链表结构和递归编程技术，以及 Turbo Pascal 中的单元技术。本章是对 Pascal 语言中的一些重点、难点的复习，是学好数据结构课程的基础。

1.1 指针类型与链表

1.1.1 指针

指针是一种简单的变量，专门用于指示某对象，即存放所指目标对象的地址。在 Pascal 语言中，指针类型说明如下：

```
type 指针类型名 = ↑目标类型;
```

说明：

(1) 一个指针只能指示某一种类型的存储单元，这种数据类型就是指针的**基类型**。基类型可以是除指针、文件外的所有类型。例如，

```
type pointer= ↑ integer;
var p1, p2: pointer;
```

定义了两个指针型变量 p1 和 p2，指针 p1 或 p2 可以指示一个整型存储单元。

(2) 和其它类型的变量一样，也可以在 **var** 区直接定义指针型变量。

(3) Pascal 语言中规定所有类型都必须先定义后使用，只有在定义指针类型时可以例外，如下定义是合法的：

```
type pointer= ↑ rec;
     rec=record
```

```
        a: integer;
        b: char
    end;
```

指针所指向的变量称为**动态变量**，在程序中需要此变量时，再为此变量申请内存空间；在不需要时，将其释放，以便合理安排内存空间。这两个功能分别用如下语句来实现。

(1) new(p): 产生一个由指针型变量 p 指向的变量，即 p↑。

(2) dispose(p): 释放由指针型变量 p 指向的变量(即 p↑)的存储空间。

可见，此处的指针型变量 p 是静态变量，而其所指向的变量 p↑是动态的。

1.1.2 链表

指针与动态变量用得最多的场合是在各种链表的算法中。链表结构如图 1.1 所示，其中：

(1) 每个框表示链表的一个元素，称为**结点**；

(2) 框的顶部示意了该存储单元的地址；

(3) 每个结点包含两个域，一个域存放整数，另一个域存放下一个结点(称为该结点的**后继结点**)的地址；

(4) 链表的第 1 个结点称为**表头**，最后 1 个结点称为**表尾**；

(5) 指向表头的指针 head 称为**头指针**(当 head 为 **nil** 时，称为**空链表**)；

(6) 在表尾结点中，由于指针域不指向任何结点，故一般放入 **nil**(画示意图时用^表示)。

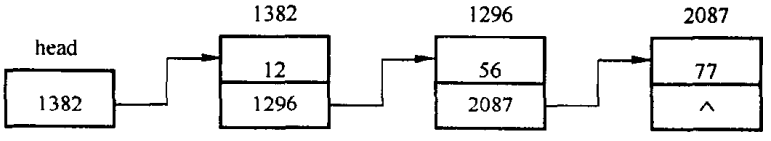

图 1.1　链表结构示意图

从图 1.1 可以看出：

(1) 链表中每个结点至少应该包含两个域，一是数据域，一是指针域。因此，每个结点都是一个记录类型，指针的基类型也是这个记录类型。可以这样定义：

```
type pointer= ↑ rec;
    rec=record
```

```
        data: integer;
        next: pointer
    end;
var head: pointer;
```

(2) 相邻结点地址不一定是连续的。整个链表通过指针来顺序访问，一旦失去一个指针值，则后面的元素将全部丢失。

(3) 与数组相比，使用链表结构时，可根据需要采用适当的操作步骤使表加长或缩短，从而使存储分配具有一定的灵活性。这是链表结构的优点。

(4) 与数组相比，数组元素的引用比较简单，直接用数组名即可。这是因为数组元素占用连续的存储单元，而引用链表元素的操作却比较复杂。

1.2 递 归 技 术

1.2.1 递归定义

如果一个子程序(过程、函数)在执行完成之前又直接或间接地调用自己，则称之为**递归子程序**(过程、函数)。若子程序在体内直接调用自身，则称为**直接递归**；若子程序通过调用其它子程序并且后者又反过来调用前者，则称为**间接递归**。

直接递归举例如下：

```
procedure p(n: integer);
  begin
    if n>0 then begin
                  p(n-1);
                  write(n:5);
                  p(n-2)
                end
  end;
```

间接递归举例如下：

```
procedure p1(n: integer);
  begin
```

```
    if n>0 then
      begin
        if odd(n) then write('*');
        p2(n-1)
      end
  end;
procedure p2(n: integer);
  begin
    if n>0 then
     begin
       if odd(n) then write('*');
       p1(n-1)
     end
  end;
```

1.2.2　递归模型

递归模型反映了一个递归问题的递归结构，例如：

$$f(0)=1 \tag{1.1}$$

$$f(n)=n*f(n-1) \qquad n>0 \tag{1.2}$$

式(1.1)给出了递归的终止条件，式(1.2)给出了 f(n)的值与 f(n-1)的值之间的关系，把式(1.1)称为**递归出口**，把式(1.2)称为**递归体**。

一般地，一个递归模型由递归出口和递归体两部分组成，前者确定递归到何时为止，后者确定递归的方式。

递归出口的一般格式为：

$$f(s_0)=m_0$$

这里的 s_0 与 m_0 均为常量，有些递归问题可能有多个递归出口。

递归体的一般格式为：

$$f(s)=g(f(s_1),\ f(s_2),\ \cdots,\ f(s_n),\ c_1,\ c_2,\ \cdots,\ c_m)$$

这里的 s 是一个递归大问题，s_1，s_2，…，s_n 为递归小问题，c_1，c_2，…，c_m 是若干个可以直接解决的问题。g 是一个非递归函数，反映了递归问题的结构。

1.2.3 递归设计

递归设计先要设计出递归模型，再将其转换成对应某语言的子程序。

从递归的执行过程考虑，要解决 f(s)，不是直接求其解，而是将其转化为计算 f(s')和一个常量 c'。求解 f(s')的方法与求解 f(s)的方法是相似的，但 f(s)是一个大问题，而 f(s')是一个较小的问题，尽管 f(s')还未完全解决，但已向目标靠近了一步，这是一个量变，如此到达递归出口时，便发生了质变，递归问题也就解决了。因此，递归设计就是要给出合理的较小问题，然后确定大问题的解与较小问题的解之间的关系，即确定递归体；最后朝此方向分解，必然有一个简单基本问题的解，并以此作为出口。由此得出递归设计的步骤如下：

(1) 对原问题 f(s)进行分析，假设出合理的较小问题 f(s')；

(2) 假设 f(s')是可解的，并在此基础上确定 f(s)的解，即给出 f(s)与 f(s')之间的关系；

(3) 确定一个特定情况(如 f(1) 或 f(0))的解，作为递归出口。

1.2.4 递归调用举例

利用函数递归调用求 n! (n≥0)的过程如下：

```
procedure ex1;
  var n: integer; y: real;
  function fac(n: integer):real;
    begin
      if n=0
        then fac:=1
        else fac :=n*fac(n-1)
    end;
  begin
    write(' input n: ') ; readln(n);
    if n<0
      then writeln(' n<0, data error! ')
      else begin
```

```
            y:=fac(n); writeln(n, '!= ', y:10:0)
        end
    end;
```

说明：

(1) 为防止溢出，将函数的类型定义为实型。

(2) 因为 fac 的形参是值形参，所以每调用一次 fac 函数就为本次调用的值形参开辟一个存储单元，这些名均为 n 的同名局部变量只在本层有效。

(3) 设在运行程序时输入 3，则执行过程如图 1.2 所示。

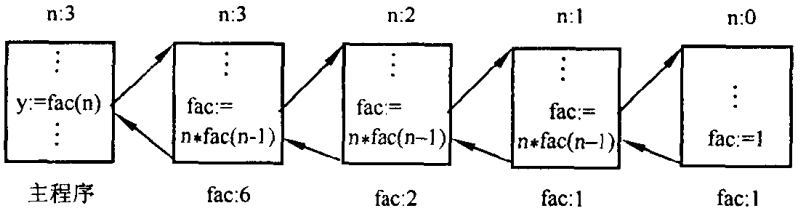

图 1.2　递归调用过程

每一个方框代表一次函数调用，方框的上部列出了本次调用时为值形参 n 建立的存储单元及值，方框的下部列出了本次调用结束传送回去的函数值，箭头表示执行的调用与返回顺序。

(4) 从本例可以看出，一个问题若要采用递归方法来解决，则必须符合两个条件：一是可以把这个问题转化成一个新问题，而新问题与原问题的解决方法相同，只是处理对象的规模不同；二是必定有一个明确的结束递归的条件。

1.3　Turbo Pascal 的单元

1.3.1　单元的基本概念

最初，设计 Pascal 语言的目的是为了教授初学者学习程序设计方法，鉴于此时程序的规模较小，因此，整个程序被组织在单个文件中。后来，随着 Pascal 语言的流行和广泛使用，愈来愈多的人开始利用 Pascal 语言编写较大规模的程序，由于程序组织在单个文件中，每当源程序产生微小变化时，都