数据结构算法设计指导

胡　学　钢

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书主要针对“数据结构”课程中具有较大灵活性和难度的算法设计技术予以阐述。全书由 6 章和 3 个附录组成。第一章介绍本书中所用的描述算法和数据结构的语言;第二章分类介绍与线性链表有关的一些算法的设计技术;第三章详细介绍与二叉树有关的典型算法的设计技术,并穿插介绍本课程中用得较频繁的递归技术;第四章以图的遍历算法为基础,以典型算法为示例介绍与图结构有关的算法的设计技术;第五章介绍与数组有关的一些典型算法的设计方法;第六章针对大多数读者深感棘手的递归技术展开了系统的讨论。附录一提供了几套模拟试卷供读者自测和复习,并在附录二中给出了评注,附录三给出了前面几章中各问题的解答。本书题材来源于长期的教学实践,书中内容按循序渐进的原则编排,采用具有较好可读性的类 PASCAL 语言,这使得本书具有较广的适用面。可作为学习数据结构以及程序设计类课程的教学参考书。

图书在版编目(CIP)数据

数据结构算法设计指导/胡学钢著. —北京: 清华大学出版社, 1999.2
ISBN 7-302-03279-3

Ⅰ.数… Ⅱ.胡… Ⅲ.数据结构-算法设计 Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字(1999)第 00097 号

出版者: 清华大学出版社(北京清华大学校内,邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者: 昌平国马印刷厂
发行者: 新华书店总店北京发行所
开 本: 787×1092 1/16 印张: 18 字数: 428 千字
版 次: 1999 年 2 月第 1 版 1999 年 2 月第 1 次印刷
书 号: ISBN 7-302-03279-3/TP·1756
印 数: 0001～5000
定 价: 21.00 元

前　言

1. 本书的出发点

“数据结构”是计算机专业一门重要的专业技术基础课程。本课程较系统地介绍了软件设计中常用的几种数据结构以及相应的存储结构和算法;介绍了常用的几种查找和排序算法,并对各部分内容从几个方面进行分析和比较,内容非常丰富。本课程的学习将为后续课程的学习以及软件设计水平的提高打下良好基础,因此本课程是计算机专业的一门关键性课程,普遍受到国内外各院校的重视。

然而,这门课程由于以下一些原因,使得学习起来相当困难:

(1) 内容丰富,学习量大,给学习带来困难;

(2) 贯穿全书的动态链表存储结构和递归技术是学习中的重点也是难点;

(3) 所用到的技术多,而在此之前的各门课程中所介绍的专业性知识又不多,因而加大了学习难度;在解答问题时也会因此而困难重重。

(4) 隐含在各部分的技术和方法丰富,也是学习中的一个重点和难点。

不少学生在解答习题尤其是算法设计题时,觉得无从下手,似乎教科书中的内容和作业题毫不相干,因此做起来特别费劲。事实上,习题中的内容和教科书中的内容是密切相关的,解答习题所需的各种技术大多可从教科书中找到,只不过其出现的形式呈多样化,因此需要仔细体会才能掌握。如果有教师指导,效果将会更好。

为了帮助学生更好地学习本课程,理解和掌握算法设计所需的技术,为整个专业的学习打好基础,本人根据学生的学习特点及自己多年的教学经验和总结,编写出了本书,希望能给学生带来一些启发。

编写本书的出发点不是要给出各种习题的答案,而是希望通过一些典型习题的解题分析及其实现的各环节来给学生一些解题示范和启发,并介绍所用到的一些技术。

另外,考虑到目前的教科书中对有些内容的讲解方法不易为学生所接受,因此在本书中也对此做了一些修订或补充。

2. 本书的内容安排

一般来说,数据结构中稍微复杂一些的算法设计中可能同时要用到多种技术和方法,如算法设计的构思方法,动态变量及链表,流程图及其变换方法,算法的编码,递归技术,与特定问题相关的技术等。某一环节用不好,都可能会影响到算法的设计。因此介绍这些方法便成了本书的主要目的。

全书的内容安排大致如下:

第一章介绍类 PASCAL 语言,较详细地描述了所用的算法描述语言。

第二到第五章分别侧重于与线性链表、二叉树和树、图结构、数组结构相关的算法的设计。其中每一节侧重于一种(或若干种)基本算法或解题方法,并以此为基础进行推广,

介绍若干算法的设计。

每个算法一般包括以下几个方面:

(1) 算法设计的分析——确定算法的形式及算法所要用到的技术;

(2) 算法设计的构思——这是算法设计的最基本要求,需要解决的问题包括算法的总体构思,模块划分,模块功能的确定及描述(包括参数的设定)等;

(3) 算法设计的实现——根据讨论绘制出流程图,并写出算法;

(4) 进一步讨论——为巩固学习效果,给出了一些与本算法及其设计过程相关的习题。

第六章较系统地介绍了本书中用得较多的递归技术。

为检验学习效果,也为了弥补本书在基本内容方面的欠缺,在附录一中给出了难度不等的九套模拟试卷(其中较容易的是 1, 2, 9; 中等难度的是 3, 4; 难度大一些的是 5, 6, 7, 8)。每套试卷中约有大小试题近 40 道,所有试卷中的试题覆盖了一般教科书中的大多数知识点,因而有助于本课程的系统复习。

附录二的试卷分析及参考答案也具有一定的特色。

为较好地理解基本知识和解答问题,附录三给出了全书习题的解答或提示。

3. 使用本书的建议

使用本书时,建议读者能按如下要求进行:

语言要求: 由于本书中算法是用类 PASCAL 语言描述的,因此,希望读者先了解 PASCAL 或类 PASCAL 语言的基本内容。

循序渐进: 由于本书是按所用的主要结构、技术和方法来分类的,每一类中按由易到难、由典型到特殊的次序排列的,因此最好能按次序阅读。但由于各算法的讲解基本上是相互独立的,因此也可以有选择地进行。

注意实践: 在阅读各题的讲解前,首先要自己动手去求解问题,在此基础上再参考书中的有关内容,并与自己的设计(或构思)相对照,最后再认真解答有关问题。实践证明,能读懂别人所设计的算法和自己能设计算法是不等价的。如果能配以《数据结构实验工具》软件一同使用,则该书的效果会更好。

如果本书能给读者带来某些启迪,那将是作者的最大欣慰。由于水平有限,书中一定存在许多不足和错误,希望能得到广大同行的指教。

作　者

1998 年 1 月于合肥

目　　录

第一章　算法描述语言简介

众所周知,“数据结构”课程的主要任务是介绍软件设计中常用的基本数据结构和相关算法的设计。本书则主要介绍其中灵活性和难度较大的算法的设计方法和技术。为此,需要选择一种合适的算法和数据描述语言,以便能方便地进行描述并使所描述的算法和数据结构易于阅读并转换成计算机语言程序。由此可见,不宜采用自然语言或严格的计算机语言。本书和大多数教科书类似,采用了一种介于高级程序设计语言和伪语言之间的语言形式,或者说是结合了两者优点的一种伪语言形式——类 PASCAL 语言,这是因为 PASCAL 语言的程序描述和数据描述能力强,并且具有较好的结构性和可读性。

下面简要介绍类 PASCAL 语言。由于类 PASCAL 语言是对标准 PASCAL 语言的一种简单扩充,因此,学过 PASCAL 语言的读者只要阅读本章第一节,了解扩充部分和放宽的一些限制就可以掌握类 PASCAL 语言。对于没有学过 PASCAL 语言的读者,通过完整阅读本章,应能对此有较全面的理解,为系统学习 PASCAL 语言打好基础。如果对此尚有不清楚的,可参阅 PASCAL 语言教材。

1.1　算法描述及语句简介

1.1.1　算法形式

如前所述,本书的主要任务是介绍算法的设计方法和技术。与大多数教科书类似,本书中所有算法均采用 PASCAL 语言中的过程和函数形式,格式分别如下:

过程形式:

```
Procedure pname(参数表);
    begin
        语句序列
    end;
```

函数形式:

```
function fname(参数表): 类型;
    begin
        语句序列
    end;
```

例 1.1: 下面两段程序分别是过程和函数形式的算法:

```
procedure abc(A, B: integer; Var C, D: integer);
  begin
      C:= A+ B; D:= A* A+ B* B
```

```
  end;
function fact(n:integer):integer;
  begin
    if n=0 then fact:=1
           else fact:=n*fact(n-1)
  end;
```

关于算法形式的补充说明:

(1) 一般来说，过程侧重于一段操作，函数侧重于一个值的计算，因此在函数说明的首行末尾要给出一个类型，以说明计算结果的类型。

(2) 每一过程或函数均用一个标识符来标识，以便在需要时可通过给出过程或函数的名称来实现调用。与 PASCAL 语言类似，pname 和 fname 分别表示过程名和函数名，如上例中，abc 为过程名，fact 为函数名。一般来说，名称是以字母开头的字母、数字和下划线序列。

(3) 为使过程或函数能在多处对不同的数据或变量进行操作，可将在各次调用中有相同的操作形式但以不同值或形式出现的量以参数形式给出，从而构成了参数表(在首行括号中)；参数表中可有若干参数，每个参数可能是值参，也可能是变参。值参和变参有以下区别(可参阅 PASCAL 语言中的“过程和函数”部分)：

形式上：变参前有关键字 var，而值参前没有。例如上面过程 abc 中，变量 A，B 为值参，而 C，D 为变参。

效果上：在调用过程前后，值参所对应的实变量的值不变，而变参所对应的变量(即实变量)在调用后的值就是在执行过程或函数体时对应变参所得到的值。简单地说，值参仅在调用时从外界得到值，而变参则当作一个变量来用——执行前从外界(实参)得到值，结束时向外界(实参)传送被改变了的值。

实参的要求：正因上述原因，在执行调用时，变参所对应的实参只能是变量，而值参则无此限制。

事实上，参数是过程和函数与外界交流的主要手段。

(4) 过程或函数的具体操作实现，即语句序列，全部放在下面的 begin 和 end 之间。语句序列中的各语句之间以“；”分隔。

为描述方便，类 PASCAL 语言对算法形式做了一些特殊放宽或补充：

(1) 在 PASCAL 语言的函数体中，至少要有一条对函数名的赋值操作，即 fname:=表达式；如例 1.1 中有两个对函数名 fact 的赋值操作。这一赋值的作用是告知系统将什么值作为该函数的本次调用所产生的结果(返回)，若有多次执行，则以最后一次的值作为结果返回。在类 PASCAL 中，还可用 return(表达式)来实现这一功能。return(表达式)的功能约定为依次执行如下两步操作：

fname:=表达式；返回(即跳出本函数的执行，返回到调用点)

(2) 在不致混淆的情况下，算法中出现的局部变量可以不做说明(PASCAL 语言则要求所有变量均要说明)。

1.1.2 语句

类PASCAL语言中的语句基本上和PASCAL语言中的语句相同，为使算法描述更方便，又做了一些补充。为便于比较，将扩充语句用* *作标注。

1. 变量赋值语句

(1) 赋值语句

格式：变量名:= 表达式;

功能：将赋值号“:=”右边表达式的值赋给左边的变量。

(2) 交换语句(* *)

格式：变量名1<= =>变量名2;

功能：互换两个变量的值。

2. 算法调用形式

过程调用当作一条语句来用，其形式为pname(实参表);

函数调用当作一表达式来用，因此只能出现在表达式可以出现的地方，如表达式中、赋值语句中，例如f:= fname(实参表)。

3. 条件语句

有两种形式的条件语句：

(1) 格式：if〈条件〉then〈语句1〉

else〈语句2〉;

功能：当〈条件〉取值为真(即布尔值true)时，执行〈语句1〉，否则执行〈语句2〉。

(2) 格式：if〈条件〉then〈语句〉;

功能：当〈条件〉取值为真(即布尔值true)时，执行〈语句〉部分的操作，否则跳过语句部分的操作。

可以将第二个条件语句作为第一个条件语句的特殊情况：语句2为“空语句”。

两个条件语句的执行流程如图1.1(a)，(b)所示。

图1.1 条件语句执行流程

条件语句中的条件是一个布尔型表达式,如布尔型变量、常量、函数、比较表达式以及由布尔运算符联结起来所构成的式子。其中的语句框只能是语法上的一条语句,当语句由多项操作构成时,需将这些语句用“语句括号”括起来,以使其变成一条复合语句,语句括号可以是 PASCAL 语言中所用的“begin...end”形式,也可以用一对方括号“[...]”表示,但考虑到后者清晰性不如前者,故本书中很少采用。以下各语句中出现的“语句”与此类似。

4. 循环语句

有三种形式的循环语句,即 while 循环、repeat 循环和 for 循环。

(1) while 循环

格式: while 〈条件〉 do
　　〈语句〉;

功能: 当〈条件〉为真时,重复执行“〈语句〉”,其中〈语句〉也称为循环体。执行流程见图 1.2。

(2) repeat/until 循环

格式: repeat 〈语句序列〉
　　until 〈条件〉;

功能: 重复执行〈语句序列〉,直到〈条件〉为真时为止,其中〈语句序列〉也称为循环体。执行流程见图 1.3。

图 1.2　while 语句执行流程　　图 1.3　repeat 语句执行流程

从流程图可以看出 repeat 循环与 while 循环的区别:

① 条件和循环体之间的执行顺序: while 语句为先判断(条件),后执行(循环体),而 repeat 语句则为先执行,后判断。由此也导致下面的差异:

while 语句的循环体可能一次也不执行,而 repeat 的循环体至少要执行一次。

② while 语句当条件成立时执行循环体,而 repeat 语句则是在条件成立时结束循环。

③ 另外,while 语句的循环体为一条语句,而 repeat 语句由于可以用 until 作为循环体的结束标志,因此其循环体可以是多条语句。

(3) for 循环

有两种格式: for 〈循环控制变量〉:=〈初值〉 to 〈终值〉 do 〈语句〉;
　　for 〈循环控制变量〉:=〈初值〉 downto 〈终值〉 do 〈语句〉;

功能: 让〈循环控制变量〉依次取从〈初值〉到〈终值〉的每一个值,每取一个值,执行

一次〈语句〉(即循环体)。所不同的是:前者(用 to)表示取值应从小到大,后者(用 downto)表示取值应从大到小,否则将不执行循环体。另外要注意:循环控制变量的类型是有序类型,如整型、字符型、布尔型等,不能是实型。

5. 情况(case)语句

前面的条件语句使得计算机可根据不同的情况采取相应的操作,因此是用得最多的一个语句。然而,如果所要区分的情况及对应的操作的数目比较多时,虽然也能用复合的条件语句来实现,但可能会带来结构上的不清晰,而情况语句(case 语句)就能以较清晰的形式实现这一功能。此处采用两种形式的情况语句。

(1) PASCAL 语言中的情况语句

格式: case 〈表达式〉of

n_1:〈语句 1〉;

n_2:〈语句 2〉;

⋮

n_k:语句 k

end;

功能:根据〈表达式〉的取值选择相应的操作:如果表达式的值为 n_1,则执行与之相对应的〈语句 1〉,否则,若为 n_2,则执行语句 2,以此类推。

由语句格式可以看出,情况语句的分支数没有明确规定,这使得程序设计人员可根据具体问题的需要来定,最后以 end 表示该语句的结束。另外,情况表达式的取值类型应为有序类型。而这一规定使得这种情况语句在某些情况下使用起来不太方便,因而引入了下面扩充的情况语句。

(2) 类 PASCAL 扩充的 case 语句(* *)

格式: case

〈条件 1〉:〈语句 1〉;

〈条件 2〉:〈语句 2〉;

⋮

〈条件 k〉:〈语句 k〉;

[else 语句 o]

end;

功能:与前面 case 语句类似,如果〈条件 1〉取值为真,则执行与之相对应的〈语句 1〉,否则,若〈条件 2〉的取值为真,则执行〈语句 2〉,以此类推,若所有条件均不满足,并且有 else 时,执行〈语句 o〉。

注意两种 case 语句的区别:后者以条件形式,即布尔表达式的形式作为选择开关,因而使用较为方便。由功能描述可知,这实际上就是复合 if 语句,因此,在用 PASCAL 语言编码时,用复合 if 语句即可简单地实现转换。

6. 出错处理语句(* *)

格式: error(‘错误信息’);
功能: 提示错误信息, 并结束算法的运行。

7. 输入输出

格式: read(变量表) —— (从键盘) 读入若干数据, 并依次赋给变量表中的变量。
格式: write(表达式表) —— 将表达式表中的各项值依次输出(到屏幕)。

8. 注释

注释是软件设计中常用的一项内容, 恰当的注释有助于软件的设计、阅读、维护和交流。因此应在学习过程中注意养成使用注释的良好习惯。注释可采用如下格式中的一种, 其中第一种形式是 PASCAL 语言所支持的。

{注释内容}, (* 注释内容*), // 注释内容//

9. 跳出循环语句(* *)

格式: exit;
功能: 跳出包含该语句的循环。

10. 最大最小函数(* *)

格式: max(n1, n2) 和 min(n1, n2)
功能: 分别返回 n1, n2 的最大值和最小值。

1.2 数据类型及其描述

软件设计中离不开对数据的组织和操作, 因此本节对此作一简要介绍。所介绍的内容都是 PASCAL 语言所支持, 并且在本课程中要用到的, 因此学过 PASCAL 语言的读者可以跳过这一节。

1.2.1 标准数据类型

PASCAL 语言中有四种标准数据类型: real, integer, char 和 boolean。

1. real(实型)

常数形式有两类, 即小数表示法和指数表示法(或科学表示法)。
小数表示法的例子: 123, 1.23, 0.123, -12.3。
指数表示法的例子: 1.23E0(= 1.23×10^{0}), 1E1(= 1×10^{1}), 1.23E-1(= 1.23×10^{-1})。
对实型数有以下一些运算: +, -, *, /。

2. integer(整型)

常数形式为数字序列，可带正负号。如 123, - 12, + 333 等。对整型数有以下一些运算: + , - , * , div(整除), mod(取模)。

注意：两种类型中除法运算的差异是 div 为整除运算，仅取整数商; mod 为取余数。如 17 div 5 的结果为 3，而 17 mod 5 的结果为 2。“ / ”只表示实数类型的除运算，即使其操作数的值为整数，也不能得到整型结果。

3. char(字符型)

常数形式为单引号内的一个字符，如 '1', 'a', 'A', ' '。字符是构成串的基本元素。字符型是有序类型，每个元素有一个序号，也就是其 ASCII 码值，如字符 'A' 的序号为 65, '1' 的序号为 49 等。每个字符 c 的序号可用序号函数 ord(c) 来求得。反之，已知序号也可求出对应的字符，这可用函数 chr 求得，如 chr(65)即为字符 'A'。

4. boolean(布尔型)

常数值仅有两个，即 false 和 true，分别代表逻辑真和假。这也是软件设计中常用的一种类型。该类型也为有序类型，其序值为 ord(false) = 0, ord(true) = 1。布尔型中的运算有:

逻辑与(and): A and B—— 只有 A, B 均为真时，结果才能为真，否则为假。

或(or): A or B —— 只有 A, B 均为假时，结果才能为假，否则为真。

非(not): not A —— A 为真时，结果为假，否则结果为真。

1.2.2 子界类型

子界类型是有序类型的子类型，通过给出其上界和下界来确定其值的范围。如 1..100，表示该类型的所有值为从 1 到 100 的所有整数，'A'..'Z' 表示从字符 'A' 到 'Z' 的所有字符，即所有大写字母。定义子界类型的说明形式如下:

type 类型名= 下界..上界;

子界类型的变量的运算与其原类型一致，所不同的是要注意值的范围。

1.2.3 构造类型

在许多情况下，一些数据对象的信息是由多个数据分量合在一起组成的，如某个学生的成绩信息是由学号、姓名和成绩等组成，一个班的某门课程的成绩表是由所有学生的成绩信息组成。这些就是下面要介绍的构造数据类型。考虑到本书的需要，主要介绍三种构造类型—— 数组、记录和集合。

在学习构造类型时，要注意以下几个方面:

(1) 类型说明或描述形式 —— 告诉计算机该构造类型的名称、是由哪些分量以什么形式组织起来的以及其说明形式。所有类型说明都在以关键字 type 开头的类型说明段中。形式为:

type 类型名= 类型描述;

(2) 定义操作对象即变量——告诉计算机要用到哪些变量以及每个变量的类型。变量说明放在 var 开头的变量说明段中，形式为:

var 变量名表: 类型名;

(3) 操作——对该类型变量所能进行的操作及其操作的表示形式。其中涉及到的一个重要方面,就是如何表示构造类型变量的各分量。

下面分别介绍各类型的有关内容。

1. 数组类型

根据数组的维数，可分为一维数组和多维数组。

(1) 一维数组

一维数组是相同类型变量的有序集合,即构成数组的各分量(元素)的类型一致,并且各分量是有序的,因此在描述时,除了要给出元素的类型(也称作基类型)外,还要给出分量的序号(即下标)分布。说明如下:

type 数组类型名= array[下标下限..下标上限] of 元素类型;

其中下标的上、下限类型为一致的有序类型,即可以是整型、字符型、布尔型等,不能是实型;元素类型几乎可以是所有类型,即除了几种标准类型及指针类型外,还可以是用户定义的构造类型,如数组、记录和集合等。

例 1.2: 下面的数组类型描述说明了 arr1 类型是由 5 个整型变量构成的数组类型,并且下标范围在 1 到 5 之间。

```
type arr1= array[1..5] of integer;
```

而变量说明 var A:arr1; 则表示数组 A 有如图 1.4 所示形式的结构。

其中,A 的所有分量的标识符依次为 A[1],A[2],A[3],A[4],A[5]。每个元素 A[i](i= 1..5)的类型为整型,因此可以当作一个普通整型变量来操作。例如可执行下面的操作:

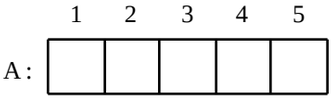

图 1.4 一维数组示例

```
A[1]:= 2* A[2]+ 5* A[3];
```

例 1.3: 下面过程求出上述数组 A 的所有元素的和并放到变参 S 中。

```
procedure sum(var S:integer);
  var i:integer;
  begin
    S:= 0;
    for i:= 1 to 5 do
      S:= S+ A[i]
  end;
```

(2) 多维数组

如前所述，数组元素可以是数组，在这种情况下，就构成了数组的数组，即多维数组。其说明与一维数组的说明一致,举例说明如下:

```
type arr1= array[1..5] of integer;
     arr2= array[0..4] of arr1;
```

此时，如果有变量说明 var A2: arr2; 则 A2 可以看作是一维数组(下标范围从 0 到 4)，A2 的每个元素又是一个一维数组(即一行，下标范围从 1 到 5)，其结构形式如图1.5所示。

在这种解释下，A[0]表示该变量中的第一行，即为一个一维数组，A[0][3]则表示该行中第三列所对应的变量，其类型为整型。

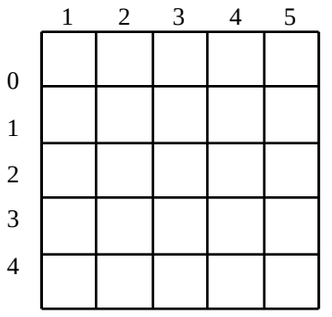

图 1.5　二维数组示例

前面的说明也可用以下的等价形式来表示:

```
type arr2= array[0..4,1..5] of integer;
```

在这一说明中的括号里，前面部分代表行下标范围，后面部分表示列下标范围。相应地，数组元素的形式为 A[i,j]，表示行下标为 i，列下标为 j 的元素。因此 A[0][3] 对应 A[0,3]。

2. 记录类型

如果组成一个对象的各分量的类型不一致，则可用记录形式来组织和表示。如前所述，学生成绩信息是由学号、学生姓名和成绩等信息组成，对此，可用如下说明:

```
type student= record
          num: string[7];
          name: string[8];
          score: 0..100;
      end;
```

该类型描述说明了记录类型 student 的组成模式：由三个分量(在此称字段) num，name 和 score 组成，其中 num 是长度为 7 的串型字段，name 是长度为 8 的串型字段，而 score 为取值介于 0 到 100 的整型字段。

更一般地，记录类型的说明形式如下:

```
type 记录类型名= record
          字段表1: 类型描述;
          字段表2: 类型描述;
              ┆
          字段表k: 类型描述;
        end;
```

该描述指出了记录类型的组成模式：所有分量及其类型。将其中在 record 和 end 之间的说明部分称为域表或字段表。

如果对前面的类型描述有变量说明 var s: student;

则 s 就是一个含有三个字段的记录型变量，s 的各字段的标识符分别为:

s.num，s.name 和 s.score

即通过在变量与字段之间加一个 '.' 来表示，因此这个点具有汉语中“的”的含义。对各字段的操作取决于其类型。例如，如果 S 的成绩不少于 90，则打印其学号和姓名可这样实现:

```
if S.score>=90 then write(S.num,' ', S.name);
```

也可将记录类型作为数组的基类型，从而构成记录数组。其操作同样取决于各自对应的类型。

例 1.4:
```
type class=array[1..30] of student;
    {以 student 为基类型构成数组类型 class,代表一个班级的成绩类型}
var C:class;                    {定义一个班级成绩表}
```

此时，C 代表一个具体班级的成绩信息，C[1], C[2], …, C[30]的类型为 student，各代表某个学生的成绩信息。C[i].score 则代表班级 C 中第 i 个学生的成绩。如果要将该班级中所有成绩在 90 以上的学生的学号和姓名都打印出来，可这样实现:

```
for i:=1 to 30 do
    if C[i].score>=90 then
        writeln(C[i].num, C[i].name);
```

3. 集合类型

PASCAL 语言是第一种支持集合运算的计算机语言。数学中，集合的元素可以是各种类型的，并且元素个数也没有限制，而在 PASCAL 语言中，对此做了一些限制：集合的元素类型(基类型)必须为有序类型，且序号只能在 0..255 之间，因此，集合的元素个数最多为 256。集合类型描述如下:

```
type 集合类型名=set of 基类型;
```

例 1.5: 下面的类型说明是错误的，因为集合类型的基数超出了规定的范围。

```
type intset=set of integer;
```

例 1.6: 对下面的集合类型及变量说明

```
type intset=set of 1..3;
var s:intset;
```

集合变量 S 可能的取值为下面 8 个之中的一个(其中用[..]表示数学中的{..}):

[ ], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]

数学中对集合的各种运算在 PASCAL 语言中也能实现，两种情况下运算符的对应关系、表达式形式和含义如下:

运算	数学符号	语言中算符	表达式形式	运算式含义
并	∪	+	A+B	由在 A、B 中出现的元素组成的集合
交	∩	*	A*B	由同在 A、B 中的元素组成的集合
差	-	-	A-B	由属于 A 而不属于 B 的元素组成的集合
属于	∈	in	e in A	判断 e 是否是 A 的元素
包含		>	A>B	判断 A 是否包含 B

例 1.7: 本例说明了集合的使用情况。

```
const maxnum=100;
type intset=set of 1..maxnum;
```

```
var S2, S3, S: intset;
    i: integer;
procedure printset(title: string; var S: intset);
  var i: integer;
  begin
        writeln; write(title+ '= [');
        for i:= 1 to maxnum do
          if i in S then
              write(i,' ');
        writeln(']');
  end;
begin
    S2:= []; S3:= [];
    for i:= 1 to maxnum div 2 do
        S2:= S2+ [2* i];
    for i:= 1 to maxnum div 3 do
        S3:= S3+ [3* i];
    printset('S2', S2);
    printset('S3', S3);
    S:= S2* S3; printset('S2* S3', S);
    S:= S2- S3; printset('S2- S3', S);
end.
```

1.2.4 指针与动态变量

前面所讨论的各种构造型结构的分量个数是确定的。然而,在许多情况下,所用结构的分量个数事先难以确定,或者说为了使程序具有较好的通用性而不能事先确定,因此需要在程序运行过程中确定,即根据具体问题的要求临时确定所要的各分量,这就是动态变量技术。

所谓动态变量,是指在程序运行过程中产生和释放的变量。使用动态变量可使程序具有较好的通用性,并能合理安排存储空间,因此是各种软件中常用的技术。这也是本课程中用得最多的一项技术。可以说,如果不掌握这一技术,则本课程的学习难以有好的效果。

如何实现动态变量?对此,一个简单的回答是:动态变量是借助于指针来实现的。因此下面先介绍 PASCAL 语言中指针的有关内容,在此基础上介绍动态变量的实现。

1. 指针

指针是一种简单类型(即非构造类型)的变量,专门用于指示某对象,即记录所指目标对象的地址。

在 PASCAL 语言中,用户定义的一种类型的指针所指向的目标的类型是一致的,即指针类型与目标类型是对应的。因此,在说明中要体现出这种关系。指针类型说明如下:

type 指针类型名= ↑目标类型;

这种说明形式较直观,很清楚地将指针类型与目标类型联系起来。目标类型可以是各种类型。另外,其中符号"↑"用键盘上的"∧"代替。

指针变量的说明和其它类型的变量的说明形式一致。