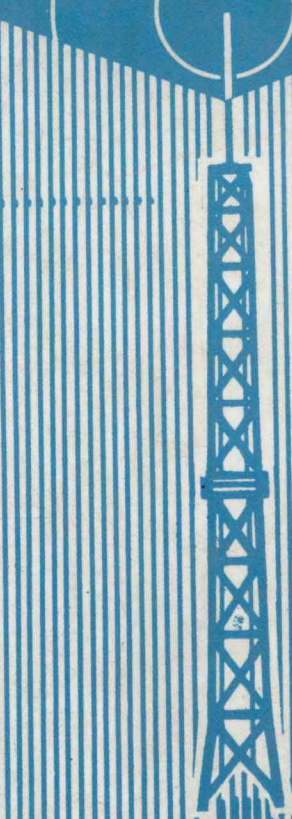
高等学校教材

数据结构

（修订版）

郭福顺　王晓芳　李莲治　编著



大连理工大学出版社

高等学校教材

数据结构

（修订版）

郭福顺　王晓芬　李莲治　编著

大连理工大学出版社

内 容 简 介

本书的前身是李莲治等编著的《数据结构》。鉴于科学技术的发展及C语言的广泛使用，现将原书作了较大修改，算法的描述主要采用C语言。全书按抽象数据类型的观点组织，共分八章。第一章是全书的基础，给出抽象数据型的定义，扼要地介绍了逐步求精的程序设计方法以及算法时间复杂度的分析方法等。第二、三、四各章是对线性表、树、二元树、图等各种主要数学模型定义相应的抽象数据型，给出各种物理表示法和有关的算法。第五、六、七章是关于数据处理技术的内容，介绍几种主要的分类和查找算法。第八章介绍文件的几种组织形式。

全书注意理论和实践的结合。内容深入浅出，因此对不同水平的计算机科技工作者都有参考价值。本书可作为计算机学科有关专业的教材或参考书。

图书在版编目(CIP)数据

数据结构/郭福顺等编著．－2版．－大连：大连理工大学出版社，1997.6
ISBN 7-5611-0101-5

Ⅰ.数… Ⅱ.郭… Ⅲ.数据结构 Ⅳ.TP311.12

中国版本图书馆CIP数据核字(97)第22283号

数 据 结 构

(修订版)

郭福顺 王晓芬 李莲治 编著

* * *

大连理工大学出版社出版发行
(大连市凌水河 邮政编码 116024)
普兰店市第一印刷厂印刷

* * *

开本：787×1092 1/16 印张：13.5 字数：303千字
1989年6月第1版 1997年6月第2版 1997年6月第6次印刷
印数：32001—36000册

* * *

责任编辑：刘晓晶 责任校对：蒋 浩
封面设计：孙宝福

* * *

ISBN 7-5611-0101-5
TP·6 定价：16.00元

出 版 说 明

根据国务院关于高等学校教材工作分工的规定，我部承担了全国高等学校、中等专业学校工科电子类专业教材编审、出版的组织工作。由于各有关院校及参与编审工作的广大教师共同努力，有关出版社的紧密配合，从1978～1985年，已编审、出版了两轮教材，正在陆续供给高等学校和中等专业学校教学使用。

为了使工科电子类专业教材能更好地适应“三个面向”的需要，贯彻“努力提高教材质量，逐步实现教材多样化，增加不同品种、不同层次、不同学术观点、不同风格、不同改革试验的教材”的精神，我部所属的七个高等学校教材编审委员会和两个中等专业学校教材编审委员会，在总结前两轮教材工作的基础上，结合教育形势的发展和教学改革的需要，制定了1986～1990年的“七五”（第三轮）教材编审出版规划。列入规划的教材、实验教材、教学参考书等近400种选题。这批教材的评选推荐和编写工作由各编委会直接组织进行。

这批教材的书稿，是从通过教学实践、师生反映较好的讲义中经院校推荐，由编审（小组）评选择优产生出来的。广大编审者、各编审委员会和有关出版社为保证教材的出版和提高教材的质量，作出了不懈的努力。

限于水平和经验，这批教材的编审、出版工作还会有缺点和不足之处，希望使用教材的单位，广大教师和同学积极提出批评建议，共同为不断提高工科电子类专业教材的质量而努力。

电子工业部教材办公室

再 版 前 言

本书是由李莲治等人编著的《数据结构》改写而成的。原书是按电子工业部制定的工科电子类专业教材1986年～1990年编审出版规划，由《计算机与自动控制》教材编审委员会计算机编审小组组织征稿、评选、推荐出版的。鉴于科学技术的发展和C语言的广泛使用，对原书作较大的修改是适宜的。修改时删除了陈旧的内容，增加了一些新的内容；对于在教学中已证明难为学生所理解的内容，作了深入浅出的解释。全书按抽象数据类型的观点组织，共分八章。其中的算法主要用C语言描述。第一章是全书的基础，给出抽象数据型的定义，扼要地介绍了逐步求精的程序设计方法以及算法时间复杂度的分析方法等。第二、三、四各章是对线性表、树、二元树、图等各种主要数学模型定义相应的抽象数据类型，给出各种物理表示法和有关算法。第五、六、七章是关于数据处理技术的内容，介绍几种主要的分类和查找算法。第八章介绍文件的几种组织形式。全书既注重原理又重视实践。各章之后附有习题。

本书是由郭福顺、王晓芬、李莲治合编，并由郭福顺统编全稿。全书各主要部分的内容，都经各编著者仔细审阅并取得一致意见。但由于编者水平有限，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 者

1997年5月

前　言

本教材按电子工业部制定的工科电子类专业教材 1986～1990 年编审出版规划，由《计算机与自动控制》教材编审委员会《计算机》编审小组组织征稿、评选、推荐出版的。

本教材由哈尔滨工业大学李莲治担任主编，成都电讯工程学院陈顺侠担任主审。

本教材是用抽象数据型的观点来组织的。全书共分十章。第一章是全书的基础，给出了抽象数据型的定义，说明数据型、数据结构和抽象数据型概念之间的区别，并扼要地介绍了逐步求精的程序设计方法。第二、三、四、五各章是对线性表、树、二元树、图等几种主要的数学模型定义相应的抽象数据型，给出各种物理表示法和算法的实现以及应用的实例。第六、七章是关于数据处理技术的内容，介绍几种主要的分类和查找算法，并对算法的效率进行扼要的分析。第八章介绍文件的几种组织形式。第九章是外部分类，介绍了磁盘各磁带文件的典型分类技术。第十章给出一个抽象数据型的应用实例。全书既注重原理又重视实现。算法用 PASCAL 语言描述。各章之后附有习题。凡是带星号 * 的习题都是较难的。

本书由郭福顺编写第一、二、十章及第四章的一部分。姜文清编写第四、五、六和七章，李莲治编写第三、八和九章并统编全稿。在本书最初（校内教材）的编写过程中，高铁军参加了部分工作，这里表示诚挚的感谢。由于编者水平有限，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　者

1988 年 3 月

目　录

第一章 绪 论

§1.1 数据结构的研究对象

计算机科学是研究信息表示和信息处理的科学。信息在计算机内是用数据表示的。直观地说，数据是用于描述客观事物的数值、字符，以及一切可以输入到计算机中并由计算机程序加以处理的符号的集合。数据的基本单位是数据元素。性质相同的数据元素的集合叫做数据对象。计算机程序通常作用在一组数据上，我们暂且称这组数据为信息表。程序在运行过程中一般要建立一些信息表，对表中的数据进行访问、加工，或在适当的时候将信息表注销。这些信息表中的数据不是杂乱无章地堆积在一起的，在表的数据元素之间存在着一定的关系，数据元素之间的这种相互关系就叫做结构。数据结构指的是数据元素之间抽象的相互关系，并不涉及数据元素的具体内容。这种结构关系将数据组织成某种形式的信息表，以便于程序进行加工。因此，信息的表示和组织形式直接影响加工程序的效率，而研究数据结构和研究算法也就很难完全分开了。

下面我们通过例子来介绍几个有关的概念，并说明数据结构的研究对象。不过，我们不想在这里对这些概念作严格而详尽地阐述，因为在以后的相应章节还会讨论到。这里只对这些概念作些说明，使读者有个直观的印象并初步了解数据结构的研究对象是什么。

信息表的一种最简单的形式是线性表，它的结构特性就是“线性”。在讨论线性表时将涉及一些与线性结构有关的问题：如哪个元素是表中的第一个元素？哪个元素是表中的最后一个元素？对于表中一个给定的元素而言，哪些元素在它之前，哪些元素在它之后？在一个线性表中总共有多少个元素？等等。可见，即使在最简单的情形，关于数据的线性结构还是有许多问题要讨论的。

更复杂一点的信息表，如多维数组、树结构等。在一般情况下，树结构是非线性的结构，它表示数据元素之间的层次关系或分支关系。

信息表是由结点组成的。每个结点由一个或多个相邻的存储单元组成。每个结点分成若干个域。每个域都有一定的名称。在最简单的情形下，一个结点可能仅由一个存储单元组成，而且可能只有一个域。

例 1.1.1 假设我们想在计算机内表示一个多项式

$$P(x)=\sum_{i=1}^{n}a_ix^i$$

由于多项式的项数可能随着对多项式的加工而增减，这就给存储分配带来一定的困难。比方说，用数组来表示多项式就不一定很合适，但用链接式线性表却是很自然的。链接式线性表就是线性表的一种具体表现形式；当用于表示多项式时，表中的每个结点表示多项式中的一个非零项 $a_ix^i(a_i\neq 0)$。如多项式

$$P(x)=x^4+6x^3+8x+3$$

可表示成图 1.1 的形式。其中,P 是指针变量;符号 Λ 表示它所在的结点是线性表的最后一个

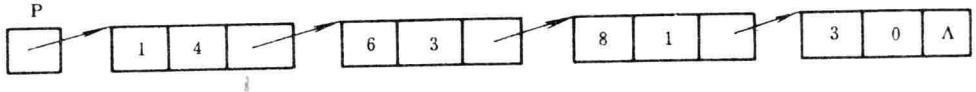


图 1.1 用链接式线性表表示多项式

结点。每个结点有三个域,分别表示对应项的系数、指数和指向下个结点的"链"。结点的地址是组成该结点的那些存储单元的首地址,这个地址也叫做指针或链。结点对应于 C 语言中的结构。假设多项式所有项的系数都是整数,则用 C 语言书写的类型说明为

```
struct Polynode {
  int coef;
  int exp;
  Polynode *link;
};
Polynode *p;
```

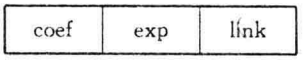


图 1.2 结点 Polynode 的格式

结点 Polynode 的格式可画成图1.2的形式。

例 1.1.2 建立人事档案是应用数据结构的一个典型例子。假设要建立一个大学的教师和学生的档案,这种结构逻辑上可画成如图1.3的形式,这就是一种

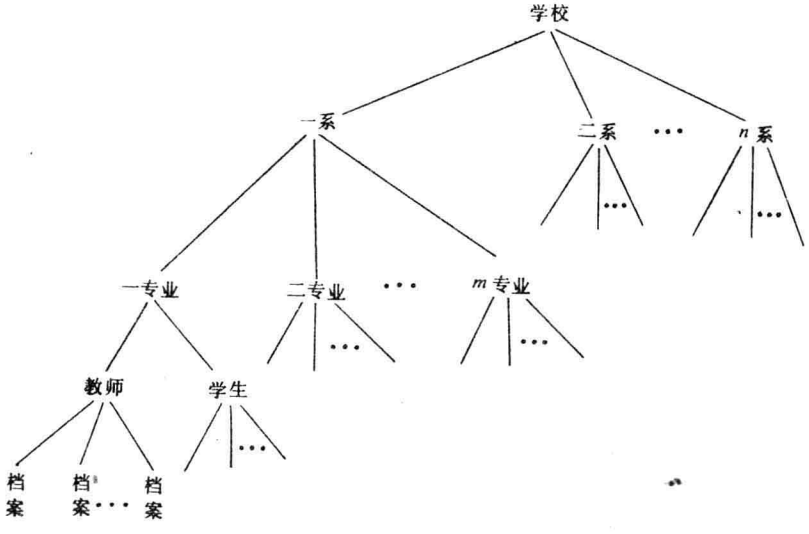


图 1.3 学校人事档案的树形结构

树结构。结构中明显地反映出层次关系或从属关系。如果我们采用这种结构来建立人事档案,就得解决它在计算机内的物理表示,以及档案的查找问题。而且许多数据结构不是静止不变的,这就是数据结构的动态特性。这种动态特性正是客观事物发展变化的反映。例如,当要增加一个系时,如何插入?一个专业被取消了,如何删除?因此,在这种结构上要定义查找、插入、删除等操作,并保证在插入和删除之后不破坏原来的结构类型。

由上述例子可以看出,数据结构主要研究数据对象的结构形式,各种数据结构的性质及其

在计算机内的表示，每种数据结构上定义的基本操作和算法；也研究算法的效率，数据结构的应用和数据分类、检索等方面的技术。

§1.2 数据结构发展概况

“数据结构”作为一门独立的课程是从1968年才开始的。在这之前，其某些内容散见于其他课程中，如“表处理语言”。1959年～1960年由 J. McCarthy 设计的 LISP 系统（表处理语言）和1962年由 D. Farber 等人设计的 SONBOL（串处理语言）系统，其数据对象的结构形式就是表结构或树结构。这些语言是以数据为中心，为处理非数值问题而设计的。而 FORTRAN、ALGOL 等算法语言则是为解决数值问题而设计的，它们侧重于以程序为中心。以程序为中心的观点侧重于建立程序；而程序则在简单数据结构上进行复杂的运算。这种观点适合于数值计算问题。另一观点把数据结构作为问题的中心（如数据库），而程序则围绕数据结构进行加工，它时而查询，时而对数据进行修改。这种观点适合于解决诸如航空订票系统等非数值问题。这类系统要求采用复杂的数据结构来描述系统的状态。可以说，程序设计以数据为中心的观点对数据结构的发展起到了推动的作用。

1968年，在美国一些大学的计算机系的教学计划中，曾把“数据结构”规定为一门课程，但对课程的内容并没有作明确的规定。当时的数据结构几乎和图论，特别和表、树的理论是同义语；后来扩充到包括网络、集合代数论、格、关系等方面，变成了现在称为“离散结构”的那些内容。然而，由于数据必须在计算机中进行处理，因此，不仅要考虑数据的数学性质，而且还必须考虑数据的存储结构。这就要求进一步扩大数据结构的内容。近年来，随着数据库系统的不断发展，数据结构课程中又增加了文件管理的内容。

数据结构在计算机科学中是一门专业基础课。在我国计算机专业教学计划中是核心课程之一。它是设计和实现编译程序、操作系统、数据库系统和其他系统程序的重要基础。

§1.3 抽象数据型

1.3.1 抽象数据型的定义

抽象数据型（Abstract Data Types，有时简写成 ADT）是程序设计语言中数据类型概念的推广。当我们在分析复杂的情况或处理复杂的事物时，经常使用抽象的思维方法，即是：舍去复杂系统中非本质的细节，只把其中某些本质的、能够反映系统重要宏观特性的东西提炼出来，构成系统的模型，并深入研究这些模型。这种抽象思维方法同样适用于软件系统的设计和研究。可以认为，一个复杂的软件系统是由一些数据结构、操作过程和控制机能所组成的。因此，在软件设计中，可以对三种不同的对象进行抽象，即过程抽象、控制抽象和数据抽象。抽象数据型就是对数据类型进一步抽象所形成的概念。这个概念对程序设计方法学和程序设计语言等都产生了深刻的影响，因此，引起人们的重视。下面通过一个例子来解释这个概念的含义，并引入概念的定义。

例 1.3.1 设有一个线性表 L。任意线性表的“类型”（或简称为“型”）记为 LIST。我们暂时不管 L 中的元素是什么东西，但可以把元素的“型”记为 elementtype。今要求编一个函数

PURGE,它以 L 为输入参数,执行结果是删除 L 中所有重复出现的元素。

```
函数 PURGE 为
void PURGE(L)
    /* 删除 L 中所有重复的元素 */
    /* 在此写上关于 L 的适当说明 */
    {
    position p, q; /* p 和 q 的类型是"位置";
                      p 指向 L 中的"当前位置";q 向前搜索以寻找与 p 相同的元素 */
(1){p=FIRST(L);
(2) while (p!=END(L)){
(3)  q=NEXT(p,L);
(4)   while(q!=END(L))
(5)     if (same (RETRIEVE(p,L), RETRIEVE(q,L)))
(6)          DELETE(q,L);
        else
(7)     q=NEXT(q,L);
(8)   p=NEXT(p,L);
      }
    }
```

PURGE 中各语句所起的作用分别是:FIRST 求 L 的第一个位置;NEXT(p,L)求 L 中位置 p 的下一个位置;END(L)用做线性表 L 的结尾标志。每当执行语句(3)(7)或(8)之前,都要测试 p 或 q 是否等于 END(L),不等才能执行;不然就会有 p=NEXT(END(L),L)或 q=NEXT(END(L),L)。它们都超出了线性表表尾之外,因而没有定义。函数 RETRIEVE(p,L)回送线性表 L 中处于位置 p 的元素。same(x,y)定义为:若 x 与 y"相同",则 same 的值为"TRUE",否则为"FALSE";其中,x 和 y 都是型为 elementtype 的元素。DELETE(q,L)表示删除 L 中位于 q 的元素。

这样编写的程序很容易读。整个程序的工作过程是这样的:变量 p 由线性表 L 的第一个位置开始向后扫描。随着程序的执行,在位置 p 前边的任何位置上,凡是与处于位置 p 的元素相同的元素均已从 L 中删除。在循环(2)~(8)的每一次执行中,q 用于扫描跟在位置 p 后面的各元素,以删除与处于位置 p 的元素相同的元素。然后 p 移向下一个位置,并重复执行上述过程,直至 p 等于 END(L)为止。

在上述程序中,线性表 L 就是一种抽象的数据,这里没有涉及 L 的具体表示,以及 L 有多长等具体性质;变量 p 和 q 也是抽象的,这里没有涉及位置是怎样表示的,是用下标呢?还是用指针?可见,函数 PURGE 不依赖于线性表的具体实现方式,无论采用什么样的数据结构来表示线性表,都不影响 PURGE 的程序编码;PURGE 和线性表打交道的唯一途径是调用线性表上的 FIRST,NEXT,DELETE 等操作。

函数 PURGE 的一般性与数据、函数的抽象性是密切相关的。例如函数 same 的抽象性就值得注意。我们曾将 same(x,y)定义为:若 x 与 y"相同",则 same 的值为"TRUE",否则为"FALSE";其中 x 和 y 都是型为 elementtype 的元素。这里,"相同"的概念是抽象的,否则就会

损害 PURGE 的一般性而使它的适用范围变窄。这是因为，若 elementtype 是实型，则当且仅当 x＝y 时，same(x,y)的值为真；然而，若 elementtype 是复合数据型，比方说是一个具有三个域的结构：

```
struct elementtype {
   int acctno;
   char name [20];
   char address [50];
};
```

则当且仅当 x. acctno＝y. acctno 时，same(x,y)的值为真。如果我们将函数 PURGE 中的条件语句写成

```
(5)  if (RETRIEVE(p,L)==RETRIEVE(q,L)
(6)          DELETE(q,L)
     else
(7)          q=NEXT(q,L);
```

则 PURGE 就不适用于 elementtype 是结构的情形了，因为这时 RETRIEVE(p,L)和 RETRIEVE(q,L)检出来的是两个结构。

在上述例子中，如果我们把线性表和线性表上定义的 FIRST，NEXT，DELETE 等操作一起看成是一个整体，就是下面要定义的抽象数据型概念的一个例子。

定义：抽象数据型是一个数学模型和在该模型上定义的操作集合的总称。

抽象数据型的例子很多。高级程序设计语言中的整型、实型和数组等都可以看成是抽象的数据类型。例如，整型数一般都定义四则运算、乘方、赋值、比较和输入、输出等操作。这些操作成为程序中对整型数进行加工处理的手段。语言的使用者只要知道这些操作的用途就可以编程序了；至于这些操作是怎样实现的，以及整型数在内存中是如何表示的，并不影响使用者所编程序的编码形式。

在例1.3.1中编制的程序有待抽象数据型的实现才能成为可工作的程序。在此例中，如果我们进一步提供 L 和 Position 的适当说明，并用函数实现线性表上的各种操作，则 PURGE 就成为一个可工作的程序了。所谓抽象数据型的实现，就是将 ADT 转换成现有程序设计语言的说明语句，加上对应于该 ADT 中每个操作的函数。换言之，抽象数据型的实现是用现有程序设计语言能够支持的适当数据结构来表示 ADT 中的数学模型，并用一组函数来实现该模型上定义的各个操作，而数据结构则利用该语言的基本数据类型和复合数据的构造机制来构成。例如，数组和结构就是 C 语言中两种主要的、复合数据的构造机制。

值得注意的是，应当把 ADT 的描述与用某种现有的程序设计语言实现 ADT 加以区别，这是大型程序设计中当前的发展趋势。为此，本书力图把 ADT 的描述与 ADT 的具体实现分成两个不同的阶段，并说明怎样去完成这两个阶段的工作。前一阶段要求我们集中地描述 ADT 的语法和语义而无须顾及其实现；后一阶段要求我们对每一种 ADT 讨论几种不同的实现方案，分析各种实现方案的利弊。当考虑 ADT 的实现时，采用什么样的数据结构来表示数学模型，要服从于能够方便和有效地实现该 ADT 上定义的大多数操作的需要。

在 ADT 的定义中还有一点值得注意。根据这个定义，如果在相同的数学模型上定义两个不同的操作集，则我们认为其代表两个不同的抽象数据型。因此，相同的数学模型以及在此数

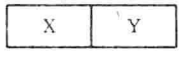
学模型上定义的相同操作，有可能在不同的抽象数据型中出现。

1.3.2 数据型、数据结构和抽象数据型

尽管术语“数据型”“数据结构”和“抽象数据型”听起来很相像，但事实上它们具有不同的意义。在程序语言中，一个变量的数据型是该变量所有可能取值的集合。例如，布尔型变量的取值只能是“真”或“假”，而不能是其他的值。基本的数据型随各程序语言而异；在C中，有整型、实型、布尔型和字符型等。利用基本数据型构造复合数据型的规则，同样随各程序语言而异；我们将在下面讲到在C中如何构造复合数据型。

抽象数据型，正如我们指出的，是一个数学模型及在该模型上定义的操作集的总称。而数据结构则是抽象数据型中数学模型的表示。我们将按照抽象数据型的观点组织和设计算法，但当用给定程序语言实现算法时，必须首先根据语言自身提供的数据型和运算符找到表示抽象数据型的具体方式。当使用数据结构来表示作为抽象数据型基础的数学模型时，我们认为结点是数据结构的基本构件；而一个结点是由若干相同或不同类型的常数或变量，按照一定的方式进行组合所形成的数据团体。给相关结点集起一个名字，并且把结点中的某个值解释为结点之间的连接信息（例如看做指向下个结点的指针），这样就建立了一个数据结构。

例如，下面的结点



由两个变量X和Y组成：X是一个整型变量，Y是指向下一个结点的指针，若没有下个结点，则令其为nil。我们把n个这种结点链起来（如图1.4），就建立了一个数据结构，其中F既可看成是该数据结构的名字，也是指向首结点的指针。

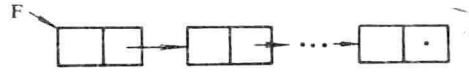


图 1.4 数据结构的建立

在C和其他程序语言中，最简单的信息聚集机制是数组。一个数组是结点的一个有序集合，其中每个结点的数据类型都相同。数组中的每个结点是相继存放的，因此，只需知道数组名（这意味着知道数组的头地址）和结点在数组中的相对位置（即下标值），即可访问数组中的每个结点。结点中的值可以是任何一种基本型或复合型的数据。

在程序语言中，把结点聚集在一起的另外一种常见的机制是结构。一个结构是由若干个称之为域的结点组合所形成的一个新结点。与数组不同，结构中每个域的类型可以互不相同，并且它们是通过名字而不是通过下标访问的。几个结构可以组成一个数组。由结构中各个域所定义的数据类型（复合型）就成为该数组中的“结点型”。例如，C说明语句表明reclist是一个具有4个元素的数组，每个元素（结点）是一个具有两个域data和next的结构：

```
struct record {
  float data;
  int next;
} reclist [4];
```

第三种聚集信息的方法是文件。文件同（一维）数组一样，是一个具有相同类型值的有序集合。然而文件中没有下标，各元素或按名访问，或按它们在文件中出现的顺序访问。利用文件聚集信息的一个优点是文件中的元素个数可以随时变化并且不受限制，这是数组所不具备的。

顺便提一下，在程序设计语言中，类型说明和变量说明是有区别的。例如，在类型说明中，

```
struct date {
  int dayofmonth;
  int month;
  int year;
  };
```

我们只规定了类型为 date 的变量的“格式”和取值范围。这时，如果没有进一步作变量说明，则实际上还不存在类型为 date 的变量；如果写出了

```
date date1, date2;
```

就说 date1和 date2是类型为 date 的变量的两个实例。这两个实例的“格式”和取值范围由 date 确定。如果在程序中有如下可执行语句

```
date1. dayofmonth=5;
date1. month=6;
date1. year=1996;
```

则 date1将表示如下日期

1996年6月5日

1.3.3 抽象数据型的规格描述

一种抽象数据型，必须首先给出规格描述，对它进行严格的定义。规格描述既是以后程序实现的根据，又可作为与程序并存的文档资料，用于程序的调试和维护。规格描述必须具有完整性、统一性和通用性。要能反映所定义的抽象数据型的全部特性，这是完整性；它应是一个前后协调的整体，不应自相矛盾，这是统一性；通用性是指所定义的抽象数据型应适用于尽量广泛的对象。同时，规格描述应尽可能不依赖于程序语言，也就是说，可以用任意的语言来实现。

如何写抽象数据型的规格描述，这方面已涌现出许多技术，其中有些已得到较普遍的认可和应用。这里不准备详细讨论这个问题。我们仅以栈为例，说明栈作为抽象数据型是怎样描述的，使读者对抽象数据型的规格描述有一基本的了解。关于栈的概念将在第二章详细叙述，但为了这里的需要，我们简单交待一下：栈是一种特殊的线性表，对栈的插入和删除等操作是在表的同一端（称为栈顶）进行的。

当描述一种抽象数据时，首先描述它的定义域；然后描述型中各个操作所起的作用。要描述型中的各个操作，就得给出语法和语义规格说明。语法规格说明指定操作的名称以及每个操作要求以何种类型的数据作为操作数。语义规格说明则给出合法表达形式（按规定的语法而言，其形式是正确的）的语义，即这种表达形式起什么作用，或它的效果是什么；换言之，语义规格说明陈述的，是我们想让程序“做什么”，但并不涉及“怎样做”的问题。因此，抽象数据型——栈的规格描述如下。

语法部分：

```
type Stack [Elementtype];
NEWSTACK ( )→Stack,
PUSH(Elementtype,Stack)→Stack,
POP(Stack)→Stack∪{UNDEFINED},
```