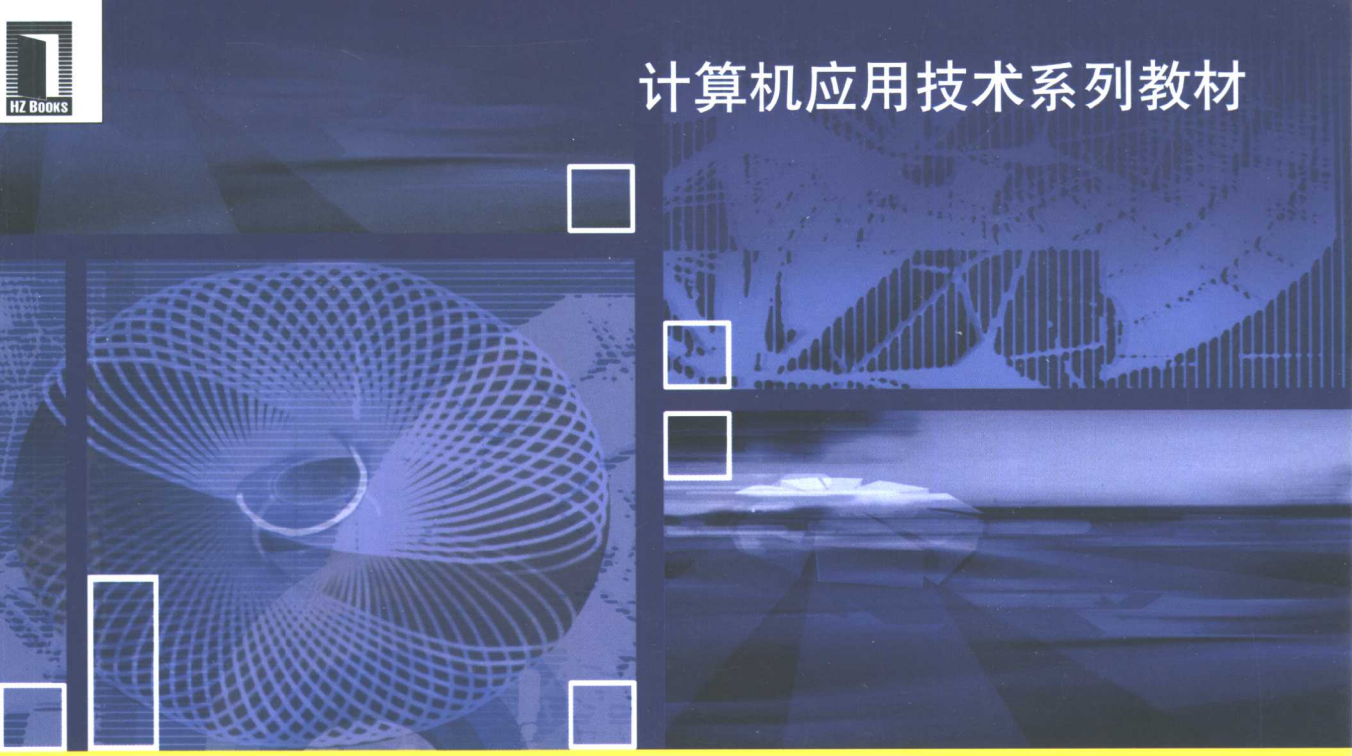
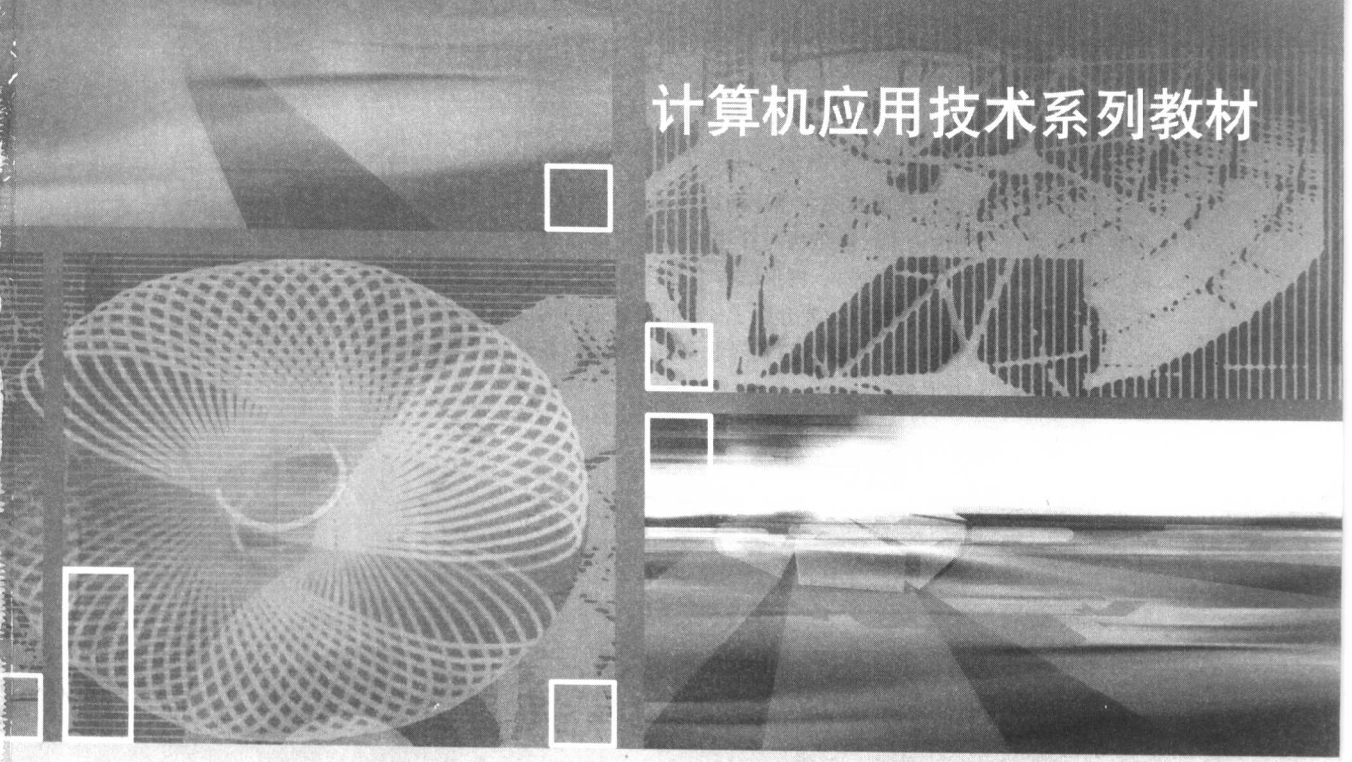


计算机应用技术系列教材

数据结构与算法教程

邹永林　周蓓　唐晓阳　杨剑勇　　编著

机械工业出版社
China Machine Press



计算机应用技术系列教材

数据结构与算法教程

邹永林　周蓓　唐晓阳　杨剑勇　　编著

机械工业出版社
China Machine Press

本书结合作者多年教学实践，循序渐进地讲述了数据结构与算法的基本概念和知识。全书共分10章，分别讨论了数据结构与算法的基础知识和表示方式，基本线性结构（线性表、栈、队列、串、数组及广义表）、树形结构、图形结构等的定义、表示和实现，排序和查找的各种方法及其实现技巧，最后简要介绍了一些扩展数据结构以及算法设计方法。

本书可作为本科、专科院校计算机专业及相关专业的教材或教学参考书。

图书在版编目（CIP）数据

数据结构与算法教程/邹永林等编著. -北京：机械工业出版社，2004.9
（计算机应用技术系列教材）
ISBN 7-111-14542-9

Ⅰ.数… Ⅱ.邹… Ⅲ.①数据结构-教材 ②算法分析-教材 Ⅳ.TP 311.12

中国版本图书馆CIP数据核字（2004）第051649号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
策划编辑：温莉芳
责任编辑：迟振春
北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行
2004年9月第1版第1次印刷
787mm×1092mm 1/16 · 17.25印张
印数：0 001-5 000册
定价：26.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

前　言

随着计算机科学技术的不断发展和应用领域的不断扩大，在许多非数值处理的应用问题中，计算机所面对的数据结构十分复杂、数据量巨大且形式多样，如何根据各类实际问题归纳、抽象出对象的数据特征及对象间的相互联系，从而选择合适的数据组织方法和存储方法，设计高效的求解算法，成为数据结构课程需要解决的最迫切的任务。

本书采用比较通俗、由浅入深和循序渐进的方式叙述了数据结构的知识。每当给出一个新的数据结构概念时，以流行的抽象数据类型（ADT）进行定义，而描述其对应的存储结构及基本操作算法时则使用C语言函数的形式，以便读者通过上机实验来理解和验证课程的具体内容和算法过程。

本书强调实用，注重理论指导下的实际可操作性，注重实际问题的解决。各章配有小结，目的在于引导读者复习该章内容；另外各章还设计了习题和实验课题，以期通过练习与实验提高学生的算法分析和设计的能力。

本书共分10章。第1章介绍数据结构和算法的基本概念，第2章、第3章和第4章介绍线性结构，第5章介绍树形结构，第6章介绍图形结构，第7章介绍排序方法，第8章介绍查找技术，第9章介绍算法设计方法，第10章介绍集合和一些扩展的数据结构。其中，前8章为数据结构课程的基本内容，不同的专业可根据需要选择讲解有关内容。另外，带“*”的章节为选学内容。

本书可作为本科、专科院校计算机科学与技术、信息管理与信息系统、计算机信息管理、计算机应用、软件工程、网络工程、电子商务等专业学生的教材或参考书。

参加本书编写的有邹永林（第1章、第2章、第10章），周蓓（第4章、第5章、第7章），唐晓阳（第3章、第6章、第8章），杨剑勇（第9章）；最后由邹永林完成本书的统稿工作。

由于作者水平有限，难免有缺点和欠妥之处，恳请读者指正。

作　者

2003年10月

目　录

第1章 概 论

自从1946年世界上诞生了第一台电子计算机以来，计算机科学与技术的发展日新月异，计算机的应用也从最初的科学计算逐步扩展，现在已渗透到数据管理、过程控制、图形图像处理等人类社会活动的各个领域；与此相对应，计算机加工处理的对象也由单纯的数值数据扩展到字符数据、图形图像数据、声音数据等相对比较复杂的、带有一定结构的、彼此存在一定关联的数据形式，这样，也给各类程序设计带来了新的问题和麻烦。为了编写出能很好地处理各种具体问题的程序，必须对所要处理的问题中包含的各种数据对象的特性及数据对象之间存在的各种关系进行深入分析和研究，从而更好地支持程序设计。由此导致“数据结构”这门学科的形成和不断发展。

1.1 引言

数据的表示是计算机科学的基础。在实际工程应用领域中，大多数计算机程序的实现目标与其说是完成运算，倒不如说是组织、存储和检索数据。从运行时间和存储空间两个方面分析，这些程序都必须合理地组织数据，以支持高效的信息处理过程。

在用计算机求解某个具体问题时，一般需要经过以下几步：1）考察具体问题，从中概括出能比较完整、准确地描述此问题的数学表示形式，即建立具体问题的数学模型；2）对数学模型进行分析研究，寻找求解此模型的方法，并用一定的形式表示求解方法，即得到合适的算法；3）借助一定的程序设计工具编写程序并进行调试，由此完成问题的求解。

在使用数学模型对具体问题进行描述时，由于问题本身的复杂性，其数学模型的表示形式也千变万化，有些问题也许可以用数学方程（组）或微积分方程（组）等相对简单的形式加以描述；但也有许多非数值数据处理问题，它们无法用较简单的数学表示形式描述出来。

1.1.1 几个例子

【例1-1】数据管理问题，如人事档案管理、财务工资管理、客户关系管理、库存物资管理、学生学籍管理等。以学生的学期课程成绩管理为例，假设某班共有48名同学，某学期共开设英语、高等数学、微机基础和哲学四门必修课程，则为了实现对该班学生的各门课程成绩的管理，可以设计如表1-1所示的表，在表中至少应包括学号、姓名、英语、高等数学、微机基础和哲学共六个数据项，分别记录每个学生的学号、姓名和各课程的成绩，其中学号项的值必须针对每个学生是惟一的。这样，可以根据表中的学号项的取值查询对应于该学号的学生的所有课程成绩；也可以通过设置查询条件如“英语课程成绩大于等于80分”来获取该课程成绩高于80分的所有学生的名单；或设置某种组合形式的查询条件如“英语、高等数学、微机基础、哲学的成绩均大于等于60分”得到该学期所有课程全部及格的学生名单，以备作为评选奖学金等活动的依据，等等。描述这种管理问题的数学模型就表现为一

张二维表，表中的记录（数据元素）按照学号依次组织，记录之间存在一对一的关系，因而这类数学模型称为“线性”的数据结构。

表 1-1 学生学期成绩表

学 号	姓 名	英 语	高等数学	微机基础	哲 学
99001	张维	82	70	83	76
99002	王大名	70	62	88	79
99003	陆丽	77	86	81	80
99004	潘志强	83	40	26	66
…	…	…	…	…	…
99047	郭世雄	75	67	56	90
99048	严春光	86	65	78	80

【例 1-2】棋类对弈游戏，如五子棋、围棋、象棋等。以最简单的井字棋对弈为例，初始状态是一个空的棋盘格局。对弈开始后，每下一步棋，则构成一个新的棋盘格局，且相对于上一个棋盘格局的可能选择可以有多种形式，因而整个对弈过程就如同图 1-1 所示的“一棵倒长的树”的形态。在这棵“树”中，从初始状态（根）至某一最终的格局（叶子）的一条路径，描述的就是一次具体的对弈过程实例。对于这类问题的数学模型，“树”状图是最简单直观的描述方式，树中的棋盘格局（数据元素）之间存在一对多的关系，这种数学模型称为“树形”的数据结构。

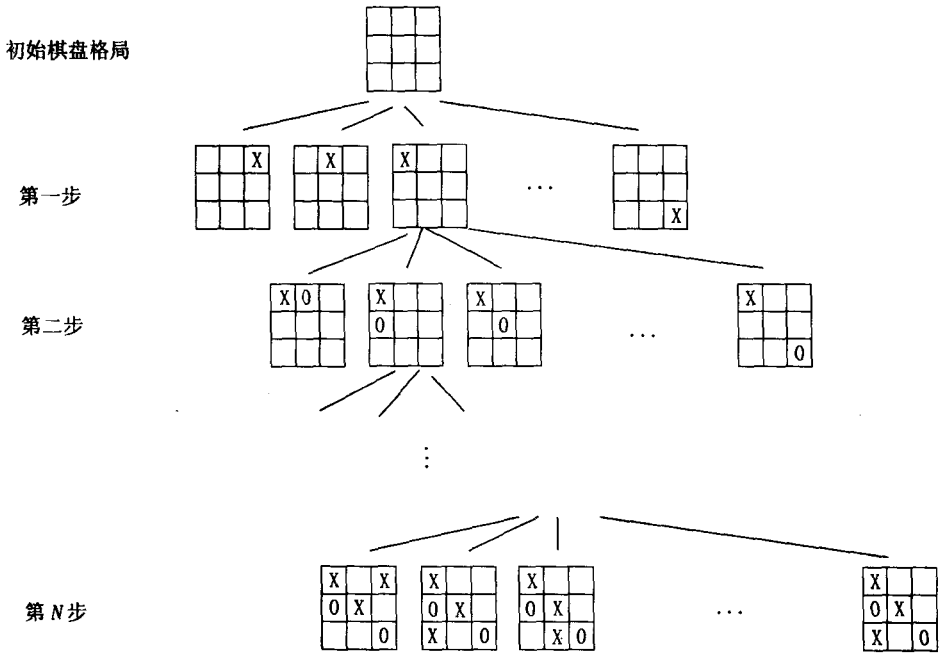


图 1-1 井字棋的对弈过程

【例 1-3】工程规划问题，如通信线路架设、工程施工计划、交通通行控制等。以专业

课程实施计划的制定为例，假设计算机应用专业在四年中共需要开设高等数学、普通物理、程序设计、离散数学、计算机原理、数据结构、编译技术和操作系统等八门学位课程，其中某些课程之间彼此构成先修课程和后续课程的关系（如图 1-2 所示）。在具体落实学年学期课程安排时，必须根据这些关系来确定各门课程的开设时间，使之符合课程教学的实际情况。在图 1-2 所示的这类规划问题中，课程（数据元素）与课程（数据元素）之间存在多对多的关系，其描述工具就是一张图，因此这种数学模型被称为“图形”的数据结构。

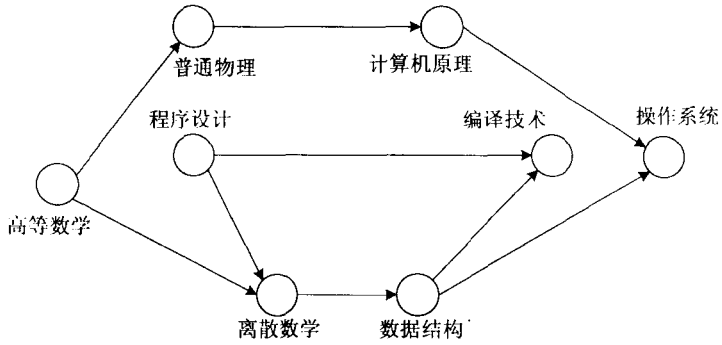


图 1-2　专业课程之间的关系

由此可知，对于非数值型数据处理问题，数学模型的描述形式不再表现为相对简单的数学方程（组），而是诸如二维表、树和图等更为复杂多变的形式。而对于这类数学模型的求解，必须通过深入研究和探索，才能正确实现。

1.1.2　数据结构的产生和发展

“数据结构”课程的产生起源于 20 世纪 60 年代。此前，随着各种计算机程序设计语言的出现，允许使用的数据类型逐渐增多，计算机需要处理的数据的组成成分日趋复杂，如 FORTRAN 语言中允许使用复数类型的数据，用两个实型数分别表示复数的实部和虚部，开始形成“结构”的雏形；COBOL 语言允许使用记录类型，它是可以用多种不同类型的数据组合而成的更复杂的结构；SNOBOL 语言能十分方便地进行串（非数值）类型数据的操作；LISP 语言中定义了带层次性的表结构，等等。对这些数据及数据之间的关系的描述、表示和操作，迫切需要通过专门、系统的分析研究来解决，从而导致了“数据结构”作为一门独立课程的形成和发展。

20 世纪 60 年代中期以后，在数据结构的发展过程中，陆续发生了几个重大的事件。首先，美国计算机界提出了信息结构（后改名为“数据结构”）的概念，1968 年美国计算机协会（ACM）颁发了建议性的计算机教学计划，首次将数据结构列为一门独立的课程；同年，著名计算机科学家 Knuth 教授发表了《计算机程序设计的技巧》第一卷《基本算法》，全面系统地论述了数据的逻辑结构和存储结构，并给出了各种典型的算法，为“数据结构”课程奠定了理论基础；20 世纪 70 年代以后，逐渐出现大型的程序和大规模的文件系统，结构化程序设计成为程序设计方法学的主要研究方向，人们对数据结构的研究越来越重视，普遍认为程序设计的实质就是对所处理的问题选择一种好的数据结构，并在此结构基础上施加一种好的算法，著名科学家 Wirth 教授的著作《算法 + 数据结构 = 程序》正是这种观点的集

中体现。此后，随着数据结构的研究不断深入，内容也逐渐固定和规范，并相继出现了各种数据结构著作和教材。

数据结构在计算机科学中是一门综合性的专业基础课，它不仅涉及计算机硬件，也与计算机软件有关。因此，可以认为数据结构是介于数学、计算机硬件和计算机软件三者之间的一门核心课程，它不仅是一般程序设计的基础，而且也是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

有关数据结构的研究仍不断发展，一方面，面向各专门领域中特殊问题的数据结构正在研究和深入，另一方面，从抽象数据类型的观点对数据结构进行探讨已成为新趋势，逐步为人们重视和应用。

1.1.3 基本概念和术语

1. 数据、数据元素和数据对象

1）数据（Data）：数据是描述客观事物的数、字符及其他所有能输入到计算机中并被计算机程序加工处理的符号（信息）的集合。通常意义下的数据形式如整数和实数等，只是数据的特例。编译程序或文字处理程序处理的数据是文件中的字符串，而多媒体处理程序处理的数据是通过特殊编码定义后的图形、图像、声音、动画等。对于计算机而言，数据的含义非常广泛，一切通过各种编码形式输入到计算机中并进行处理的对象均可归属到数据的范畴。

2）数据元素（Data Element）：数据元素是数据的基本单位，是数据集合中的个体。通常，在计算机程序中它作为一个整体进行考察和处理。所谓基本单位，是指其大小可变，可根据描述数据个体的性质的需要确定。每个数据元素既可以只包含一个数据项（Data Item 或 Field），也可以由若干数据项组合而成。数据项是构成数据元素且具有独立现实意义的、不可分割的最小单位。

3）数据对象（Data Object）：数据对象是性质相同的数据元素的集合，是数据集合中的一个子集。如整型数据对象是集合 $\{0, \pm1, \pm2, \cdots\}$，字母字符数据对象是集合 $\{A, B, \cdots, Z, a, b, \cdots, z\}$，等等。

2. 数据结构

数据结构是相互之间存在一种或多种特定关系的数据元素的集合。在各类实际应用问题中，数据元素之间总是存在着各种关系，描述数据元素之间关系的方法称为结构。通常，可根据数据元素之间所存在的关系的不同特征，用四类基本结构予以描述：1）集合：指结构中的数据元素之间只存在“同属一个集合”的关系；2）线性结构：指结构中的数据元素之间存在“一对一”的关系；3）树形结构：指结构中的数据元素之间存在“一对多”的关系；4）图形结构：指结构中的数据元素之间存在“多对多”的关系。图 1-3 为上述四类基本结构示意图。

1）数据结构的形式定义：借助集合论，数据结构可描述为一个二元组

$$\text{Data\_ Structure} = (D, S) \tag{1-1}$$

其中，D 是数据元素的有限集合；S 是 D 集上关系的有限集合。可通过以下例子加以理解。

【例 1-4】复数的数据结构表示。

在计算机科学中，复数可定义为如下形式：

$$\text{Complex} = (C,R) \tag{1-2}$$

其中，C 是包含两个实数的集合 $\{r_1, r_2\}$；R 是定义在集合 C 上的一种关系 $\{<r_1, r_2>\}$，其中有序对 $<r_1, r_2>$ 分别代表复数的实部和虚部。

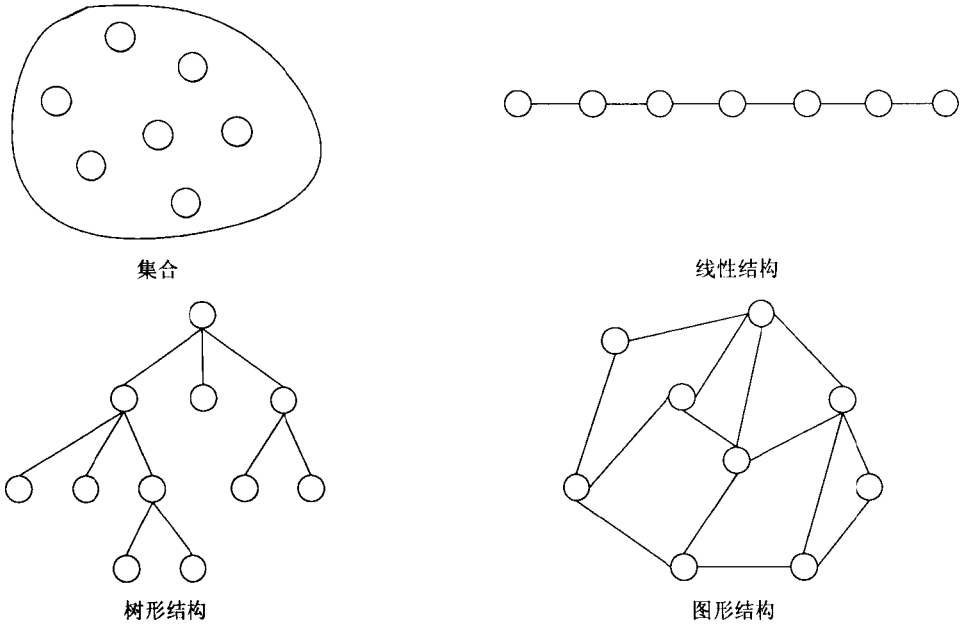


图 1-3　四类基本结构示意图

【例 1-5】某城市电话号码簿的数据结构表示。

某城市的电话号码簿可定义为：

$$\text{Telephone\_Book} = (T,R) \tag{1-3}$$

其中，$T=\{(u_1,t_1),(u_2,t_2),\cdots,(u_n,t_n)\}$，$n$ 表示用户个数，u_i 表示用户名，t_i 表示对应的电话号码；$R=\{<N_i,N_{i+1}> \mid N_i,N_{i+1}\in T,i=1,2,\cdots,n-1\}$，$N_i=(u_i,t_i)$。

【例 1-6】成绩管理系统中某班级学生的学期成绩表的数据结构表示。

某班级学生的学期成绩表可定义为：

$$\text{Score\_Table} = (D,R) \tag{1-4}$$

其中，$D=\{(r_1,n_1,s_{11},\cdots,s_{1k}),(r_2,n_2,s_{21},\cdots,s_{2k}),\cdots,(r_m,n_m,s_{m1},\cdots,s_{mk})\}$，$m$ 为该班学生数，k 为学期课程数，r_i 为学生序号，n_i 为学生姓名，s_{i1} 为第一门课程成绩，s_{ik} 为第 k 门课程成绩；$R=\{<N_i,N_{i+1}> \mid N_i,N_{i+1}\in T,i=1,2,\cdots,m-1\}$，$N_i=(r_i,n_i,s_{i1},\cdots,s_{ik})$。

数据结构的形式定义是从数学角度对操作对象的描述，也就是说，是通过抽象得到的数学模型。这种数学模型描述的是数据元素之间的逻辑关系，因此，又称为数据的逻辑结构。相对于数据的逻辑结构，为了在计算机中实现对应的操作，还必须讨论它在计算机中的表示方式。

2）数据的存储结构（物理结构）：数据的逻辑结构在计算机中的表示（映像）称为数据的存储结构或物理结构。这个映像包括数据元素的表示和数据元素之间关系的表示两个方面。

在计算机中，数据信息的表示（存储）单位有：位（Bit）、字节（Byte）、千字节（KB）、兆字节（MB）等，不同类型的数据在存储时需要的空间大小是不同的，一般地，一个整型数据需要2字节，一个字符型数据需要1字节，一个实型数据则需要4字节，等等；数据的逻辑结构中包含的数据元素在计算机中的存储形式称为元素（Element）或结点（Node），若数据元素由多个数据项组成，则每个数据项在元素或结点中的对应部分称为域（Field），根据数据项中的数据的类型，分为数据域（Data Field，存放普通类型的数据）和链域（Link Field，存放指针型数据，也称指针域）。因此，所谓元素或结点，是指数据元素在计算机中的映像（存储结构）。

数据元素之间的关系在计算机中表示时，通常可采用顺序映像和非顺序映像两种不同的方式实现。所谓顺序映像，是指数据元素之间的逻辑关系借助对应结点的存储单元的相对位置关系来表示，也就是说，逻辑关系相邻的数据元素用物理位置相邻的结点顺序表示；而非顺序映像是指数据元素之间的逻辑关系借助对应结点中的链域（表示结点存储单元地址的指针）中的数据来具体描述，与各结点的实际存储位置无关。

由上可知，数据的逻辑结构中描述的数据元素之间的关系与数据的存储结构中结点的物理位置之间没有必然的对应关系，在逻辑结构基础上定义的基本操作依赖于数据的存储结构，因此，研究数据结构不能不涉及有关的基本操作及实现方法。

3）基本操作及实现：对于任何一种数据结构，当其包含的数据元素之间的关系确定以后，对应的存储结构也可以根据具体问题的需要而确定。若求解过程需要涉及结构中元素的值或存储位置的改变、元素的增加或删除等，这些对原来存储结构的修改行为，就是所谓的基本操作。一般来说，在定义某种数据结构时，所对应的各种基本操作也同时予以定义；将它表示成物理结构时，也同时给出对应的基本操作函数。

因而，完整的数据结构概念可认为是由数据的逻辑结构、存储（物理）结构及基本操作集三个部分组成，可用以下形式加以描述：

$$\text{Data\_Structure} = (D, S, P) \tag{1-5}$$

其中，D 是数据元素的有限集合；S 是 D 集上关系的有限集合，P 是对 D 的基本操作集。

3. 数据类型

数据类型（Data Type）是程序设计语言中对于给定变量的所有可能取值的集合。例如，在典型的16位计算机中，整型变量允许的取值范围一般为 -32 768 ~ +32 767 之间的整数；而逻辑型变量的取值则是“真”或“假”之一。

每一种程序设计语言都有一组固有的或基本的数据类型。对于 FORTRAN 语言，基本数据类型有：INTEGER、REAL、LOGICAL、CHARACTER 和 COMPLEX 等；对于 C 语言，基本数据类型有：int、float、long int、double、char、enum 和指针等；而许多现代的程序设计语言中允许定义新的类型，这些新的类型既可以是对基本数据类型的限制，也可以是若干基本类型的组合形式。

如果某个对象是仅由单值构成的形式组成的类型，则称为原子类型，如整型、字符型等；相反，若是由一组值构成的形式组成的类型，则称为结构类型或组合类型，可进一步分解为若干成分，每一个成分既可以是原子类型，也可以是另一结构类型。一般来说，程序设计语言中提供的基本操作都是在原子类型上进行的。

从某种意义上说，数据类型可理解成在程序设计语言中实现了的数据结构。

4. 抽象数据类型

抽象数据类型（Abstract Data Type，ADT）是一种数据类型及在这种数据类型上定义的一组操作。抽象数据类型不仅包括数据类型的定义，同时也为这种类型说明了一个有效的操作集合。虽然从某种意义上看，抽象数据类型与数据类型在本质上相通，例如，在各种计算机上都具有“整型”数据类型，它所定义的数学特性是一致的，然而，由于它在不同处理器中的实现方法可以有差别，因此，它就是一种抽象数据类型。但是，在抽象数据类型层次，数据类型的范畴更广，它不仅可以包括在处理器中已经定义并实现的数据类型，也可以包括用户自己设计的数据类型。“抽象”的意义就在于数据类型的数学抽象特性。

利用抽象数据类型来描述数据结构，有助于我们在设计一个软件系统时，不必首先考虑其中包含的数据对象及操作在不同处理器中的表示和实现细节，而是在构成软件系统的每个相对独立的模块上定义一组数据和对应的操作，把这些数据的表示和操作的细节留在模块内部解决，在更高的层面进行软件的分析和设计，从而提高软件的整体性能和复用率。

和数据结构的形式定义相对应，抽象数据类型也可以用一个三元组描述：

$$\text{Abstract Data Type} = (D,S,P) \tag{1-6}$$

其中，D 是数据元素的有限集合；S 是 D 集上关系的有限集合，P 是对 D 的基本操作集。本书约定，抽象数据类型的定义格式采用如下形式：

```
ADT 抽象数据类型名 {
    数据对象：<数据对象的定义>
    数据关系：<数据关系的定义>
    基本操作：<基本操作的定义>
} ADT 抽象数据类型名
```

其中，数据对象和数据关系的定义用伪码描述，基本操作的定义采用以下格式：

```
基本操作名（参数表）
    初始条件：<初始条件的描述>
    操作结果：<操作结果的描述>
```

其中，“初始条件”描述操作执行之前数据结构和参数应满足的条件，若不满足，则操作失败，并返回相应的出错信息，且初始条件为空时，可省略；“操作结果”则表示操作正常完成之后数据结构的变化状况和应返回的结果。

抽象数据类型可以借助处理器中已存在的数据类型来表示和实现。也就是说，可以通过某种具体的高级程序设计语言的固有数据类型和自定义数据类型，并使用过程和函数来表示和实现抽象数据类型。在所有的算法语言中，C 语言是一种数据类型丰富、使用最广泛的高级程序设计语言，尽管 C 语言不直接支持 ADT，但鉴于目前国内的现实，本书采用 C 语言阐述各种数据类型的 ADT 说明、表示和实现。

下面以复数为例，给出一个完整的抽象数据类型的说明、表示和实现方法。

在计算机科学中，复数可定义为一种简单的抽象数据类型：

$$\text{Complex} = (C,S,P)$$

其中，C 是含有两个实数的集合 $\{r_1,r_2\}$；S 是定义在集合 C 上的一种关系 $\{<r_1,r_2>\}$，有序对 $<r_1,r_2>$ 表示 r_1 是复数的实部，r_2 是复数的虚部；P 是对 C 的基本操作集。

1）说明部分：

```
ADT Complex {
```

数据对象：$C=\{(r_1,r_2)\mid r_1,r_2\in R,R$ 是实数集$\}$

数据关系：$S=\{<r_1,r_2>\mid <r_1,r_2>$ 表示 r_1 是复数的实部，r_2 是复数的虚部；每个元素表示定义在 C 上的一个孤立的值$\}$

基本操作：

void Create(float x,float y,Complex &c)

生成一个复数 c,$c=x+iy$

Complex Add(Complex $c1$,Complex $c2$)

求两个复数 $c1$ 和 $c2$ 的和,sum = $(x1+x2)+i(y1+y2)$

Complex Sub(Complex $c1$,Complex $c2$)

求两个复数 $c1$ 和 $c2$ 的差,difference = $(x1+x2)-i(y1+y2)$

Complex Mult(Complex $c1$,Complex $c2$)

求两个复数 $c1$ 和 $c2$ 的积,product = $(x1*x2-y1*y2)+i(y1*x2+y2*x1)$

float Realpart(Complex c)

取复数 c 的实部,realpart = x

float Imagepart(Complex c)

取复数 c 的虚部,imagepart = y

}ADT Complex

2）表示部分：

```
typedef struct Complex{        /*复数类型*/
        float realpart;        /*实部*/
        float imagepart;       /*虚部*/
}Complex;
```

3）实现部分：

```
void Create(float x, float y, Complex &c)
    {c.realpart=x;
    c.imagepart=y;
    }
Complex Add(Complex c1, Complex c2)
    {/*求两个复数 c1 和 c2 的和 sum*/
    Complex sum;
    sum.realpart=c1.realpart+c2.realpart;
    sum.imagepart=c1.imagepart+c2.imagepart;
    return sum;
    }
Complex Sub(Complex c1, Complex c2)
    {/*求两个复数 c1 和 c2 的差 difference*/
    Complex difference;
    difference.realpart=c1.realpart-c2.realpart;
    difference.imagepart=c1.imagepart-c2.imagepart;
    return difference;
    }
Complex Mult(Complex c1, Complex c2)
    {/*求两个复数 c1 和 c2 的积 product*/
    Complex product;
    product.realpart=c1.realpart*c2.realpart-c1.imagepart*c2.imagepart;
    product.imagepart=c1.imagepart*c2.realpart+c2.imagepart*c1.realpart;
    return product;
```

```
    }
float Realpart(Complex c)
    {/*取复数 c = x + iy 的实部 x */
    return c.realpart;
    }
float Imagepart(Complex c)
    {/*取复数 c = x + iy 的虚部 y */
    return c.imagepart;
    }
```

通过上述实例，我们对抽象数据类型的概念能得到更完整、更准确的理解。

1.2　问题、算法和程序

在工程应用实践中，程序设计人员经常需要涉及和处理各种问题、算法和计算机程序。这是三个不同的概念。

1.2.1　问题

从一般意义上理解，问题（Problem）就是需要完成的任务、需要解决的工作。解决问题的过程通常表现为对应一组初始条件的输入有一组相应的输出结果。在问题的定义中不能包含有关怎样解决问题的限制，只有在准确定义并完全理解问题后，才能研究问题的解决方法。但是，在问题的定义中应该包括对任何可行方案所需资源的限制。对于使用计算机解决的任何问题，总有一些直接和间接的限制，如计算机的可用存储及运算空间、所规定的运算时间范围等。

从数学角度看，一般可把问题与函数等同考虑。在数学中，函数是输入和输出之间的一种映射关系，函数的输入可以是一个值或一组值，这些值组成整个函数的（输入）参数。不同的输入可以产生不同的输出结果，然而对于给定的输入，函数每次计算得到的结果必须相同。有人可能会想，用计算机程序表示的日期函数，对于不同的计算机，在不同的日子运行所得到的结果是不同的。但是，如果初始设置相同，在任意一个确定的日子，正确运行日期函数所得到的结果仍然是惟一的，这并不矛盾。

1.2.2　算法

通常，算法（Algorithm）是指解决问题的一种方法或一个过程。如果把问题看成函数，则算法能把输入转化成输出。一个问题可以有多种算法，一个给定的算法可以用来解决一个特定的问题。了解对于同一问题的多种求解算法，有助于提高对于解决同一问题中所包含的不同输入特性的运行效率，加深对算法的理解。

在数据结构中，算法是对特定问题求解步骤的一种描述，是指令的有限序列。它具有以下特性：

1）有穷性：算法必须由有限步组成，在有限的时间内执行结束。其中“有限”的含义是指算法的描述在篇幅上有穷，可经过一定的运行时间得到结果，算法的运行方向是逐步趋于结束的。

2）确定性：算法中所描述的每一步都有明确的含义，表示算法执行过程中的实际动作，完成算法所规定的具体任务，不能存在理解上或执行中的歧义。

3）可行性：算法所描述的行为对于使用该算法的人或计算机必须是可读的、可理解的、可执行的，也就是说，可以根据算法的描述，完成对问题的求解，得到正确的结果。

4）输入：算法必须有零个或多个输入，表示某个问题所对应的初始状态或条件。

5）输出：算法必须有一个或多个输出，表示对某个问题的求解结果。

1.2.3 程序

通常，计算机程序（Program）可认为是对于算法运用某种程序设计语言的具体实现。事实上，在计算机应用领域，几乎所有的算法最终都以计算机程序的形式表示和描述，但应当注意，程序和算法是两个不同的概念。

一个计算机程序可以描述一个特定算法的实现过程，也可以包含由多个特定算法描述的许多任务的综合体。一个特定的算法可以用一个计算机程序描述和实现，但是，一个计算机程序描述和表示的却不一定是一个算法。

以操作系统为例，任何操作系统都是程序，它包含了许多具体的操作。在操作系统的运行过程中，每当响应用户请求时，所对应的操作任务可以看成是一个单独的问题，由特定的算法实现其求解，得到相应的输出结果后，该特定问题的算法便结束，但是，操作系统的运行并未就此结束，仍在等待新的任务。因而，虽然操作系统是程序，但不能说它是算法。

1.3 算法描述和分析

前面已经介绍了算法的概念。对于算法，还有两个问题必须解决：1）如何将一个特定的问题表示为一个算法，用什么方式将算法正确和完整地加以描述；2）所选择的算法的运行效率怎样，如何评价一个算法的效率。

1.3.1 算法描述

对于任何实际问题，经过分析可以得到计算机能解决的数学模型，然后再选择合适的数据结构组织数据，选择恰当的算法进行计算。可以采用多种方法将一个算法的求解过程和步骤完整、准确地描述出来。一般地，描述算法的常用方法有：自然语言表示法、伪码表示法、流程图表示法、N-S 图表示法、程序设计语言表示法等。以计算 5！为例，采用上述各种方法进行描述的过程大致如下。

1. *自然语言表示法*

用自然语言来描述算法，就是用文字形式将指定算法的求解步骤正确地加以表述。具体形式可以是：

设：T 表示被乘数，I 表示乘数，计算结果存放到被乘数 T 中。则

Step1：使 $T=1$；

Step2：使 $I=2$；

Step3：使 $T\times I$，结果存放到 T 中，即 $T\times I\rightarrow T$；

Step4：使 I 的值加 1，即 $I+1\rightarrow I$；

Step5：如果 I 不大于 5，返回继续执行 Step3、Step4、Step5；否则，算法结束。

2. *伪码表示法*

用伪码来描述算法，就是用与某种程序设计语言的语法、结构和规范相类似的形式对算