清华大学计算机系列教材

数据结构

（第二版）

严蔚敏　吴伟民　编著

清华大学出版社

数 据 结 构

（第二版）

严蔚敏　吴伟民　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

数据结构(第二版)是 87 年出版的原书的修订版。修订版在保持原书基本框架和特色的基础上对主要各章如:第一、二、三、四、六及九章等作了增删和修改。

本书系统地介绍了各种类型的数据结构和查找、排序的各种方法。对每一种数据结构,除了详细阐述其基本概念和具体实现外,并尽可能对每种操作给出类 PASCAL 的算法,对查找和排序的各种算法,还着重在时间上作出定量或定性的分析比较,最后一章讨论文件的各种组织方法。

本书概念清楚,内容丰富,并有相配套的《数据结构题集》,既便于教学,又便于自学。

本书可作为大专院校计算机专业和计算机应用专业的教材,也可供从事计算机工程与应用工作的科技工作者参考。

数 据 结 构
(第二版)
严蔚敏 吴伟民 编著
责任编辑 贾仲良
☆
清华大学出版社出版
北京 清华园
化学工业出版社印刷厂印刷
新华书店总店科技发行所发行
☆
开本:787×1092 1/16 印张:22 字数:525 千字
1992 年 6 月第 2 版 1999 年 6 月第 19 次印刷
印数:942001～973000
ISBN 7-302-00984-8/TP・363
定价:21.00 元

第二版前言

本书对87年版的原书进行了改写，其中较重大的修改为：

在第一章中增加了说明数据结构和数据类型的例子，并引入“抽象数据类型”的概念及其实现方法；

在第二章中，删去了“等价问题的求解”一节，增加了以有序表表示集合的应用例子；

在第三章中，删去了“多个顺序栈共享存储空间”的内容，并对“递归过程及其实现”作了重大修改；

对第四章，重新组织了“串的存储结构”一节的内容，并增加了“建立词索引表”的应用例子；

在第六章删去了“博弈树”一节，而以“回溯法”作为树的遍历的应用例子；

对第九章，从“静态查找表”和“动态查找表”这两个抽象数据类型的不同表示和实现的角度来讨论各种查找方法。

在每一章的应用例子中，尽可能从抽象数据类型角度加以讨论。其它如对二叉树线索化的处理等也作了些改动。

由于水平所限，可能对一些新的知识理解不透，书中难免存在缺点和疏漏，敬请广大读者予以批评指正。

作　者

1991年10月

前　言

本书详细介绍了线性表、栈和队列、串、数组和广义表、树和二叉树以及图等几种基本类型的数据结构，以及在程序设计中经常遇到的两个问题——查找和排序。全书共分十二章。在第一章中首先以三个非数值性的程序设计问题为例概括地介绍了"数据结构"研究的对象，并综述了数据、数据结构和数据类型等基本概念，然后对书中描述算法所用语言以及算法的度量作了概要说明；在第二章至第七章分别讨论了上述几种数据结构，对每一种结构力求从数据元素之间固有的关系出发给出恰当的描述，同时，为了明确表示数据结构在计算机中的表示，本书中采用了类似于 PASCAL 语言的类型说明来定义存储结构，并在讨论基本运算的基础上给出一些应用例子；第八章综合介绍操作系统和编译程序中涉及的动态存储管理的基本技术；第九章至第十一章讨论查找和排序，在这三章中，除了介绍各种算法之外，还着重从时间上作定性或定量的分析和比较；第十二章讨论了文件的物理结构。

本书可作为计算机系本科学生的教材，讲授学时为 60～80。在学时少的情况下，讲授教师可根据本校学生的情况酌情删去书中某些内容，本书在目录页中给出参考性的意见（即建议删去带 * * 的章节）；对侧重于计算机应用专业的学生，还可删去第五、八和十一章的内容。为了便于读者自学，本书在文字叙述上力求做到语言通俗、简明易懂，特别是对第二章至第七章的内容，解释颇为详细，其中绝大多数的运算都给出了类 PASCAL 语言描述的算法，只要稍加修改，便可变成能上机执行的 PASCAL 程序。

从课程性质上讲，《数据结构》是一门专业技术基础课。它的教学要求是：学会分析研究计算机加工的数据对象的特性，以便选择适当的数据结构和存储结构以及相应的算法，并初步掌握算法的时间分析和空间分析的技巧。另一方面，学习本课程的过程也是进行复杂程序设计的训练过程，要求学生书写的程序结构清楚、正确易读。因此，整个教学过程中，习题和上机实习是两个不可缺少的环节，与本书各章配套的习题和实习题集中一起另编成册（《数据结构题集》，已由清华大学出版社出版）。

本书也可供从事计算机应用等工作的工程与科技人员参考。只需掌握编制程序的基本技术便可学习本书，若具有离散数学和概率论的知识，则对书中某些内容更易理解。

本书由沈阳工业学院的曹素芬同志和北京计算机学院的陈文博同志审阅了部分初稿，并提出许多宝贵意见。全书由金慧芬同志帮助抄写，在此一并表示衷心感谢。

有本书作者参加编写并于 1980 年出版的统编教材《数据结构》曾得到许多读者的关切，并来信指正书中错误，在此向读者致歉意和感谢。由于水平所限，本书中仍难免出现错误和缺点，恳切希望继续得到广大读者特别是讲授此课程的老师们的批评和指正。

作　者

1986 年 4 月

第一章 绪 论

自 1946 年第一台计算机问世以来，计算机产业的飞速发展已远远超出人们对它的预料，在某些生产线上，甚至几秒钟就能生产出一台微型计算机，产量猛增，价格低廉，这就使得它的应用范围迅速扩展。如今，计算机已深入到人类社会的各个领域。计算机的应用已不再局限于科学计算，而更多地用于控制、管理及数据处理等非数值计算的处理工作。与此相应，计算机加工处理的对象由纯粹的数值发展到字符、表格和图象等各种具有一定结构的数据，这就给程序设计带来一些新的问题。为了编写出一个“好”的程序，必须分析待处理的对象的特性以及各处理对象之间存在的关系。这就是“数据结构”这门学科形成和发展的背景。

1.1 什么是数据结构

一般来说，用计算机解决一个具体问题时，大致需要经过下列几个步骤：首先要从具体问题抽象出一个适当的数学模型，然后设计一个解此数学模型的算法，最后编出程序、进行测试、调整直至得到最终解答。寻求数学模型的实质是分析问题，从中提取操作的对象，并找出这些操作对象之间含有的关系，然后用数学的语言加以描述。例如，求解梁架结构中应力的数学模型为线性方程组；预报人口增长情况的数学模型为微分方程。然而，更多的非数值计算问题无法用数学方程加以描述。下面请看三个例子。

例 1-1 图书馆的书目检索系统自动化问题。当你想借阅一本参考书但不知道书库中是否有的时候；或者，当你想找某一方面的参考书而不知图书馆内有哪些这方面的书时，你都需要到图书馆去查阅图书目录卡片。在图书馆内有各种名目的卡片：有按书名编

001	高等数学	樊映川	S01	…
002	理论力学	罗远祥	L01	…
003	高等数学	华罗庚	S01	…
004	线性代数	栾汝书	S02	…
⋮	⋮	⋮	⋮	⋮

高等数学	001,003,…
理论力学	002,…
线性代数	004,…
⋮	

樊映川	001,…
华罗庚	003,…
栾汝书	004,…
⋮	

L	002,…
S	001,003,…
⋮	

图 1.1 图书目录文件示例

排的、有按作者编排的、还有按分类编排的，等等。若利用计算机实现自动检索，则计算机处理的对象便是这些目录卡片上的书目信息。列在一张卡片上的一本书的书目信息可由登录号、书名、作者名、分类号、出版单位和出版时间等若干项组成，每一本书都有唯一的一个登录号，但不同的书目之间可能有相同的书名、或者有相同的作者名或者有相同的分类号。由此，在书目自动检索系统中可以建立一张按登录号顺序排列的书目文件和三张分别按书名、作者名和分类号顺序排列的索引表，如图 1.1 所示。由这四张表构成的文件便是书目自动检索的数学模型，计算机的主要操作便是按照某个特定要求（如给定书名）对书目文件进行查询。诸如此类的还有查号系统自动化、仓库帐目管理等。在这类文档管理的数学模型中，计算机处理的对象之间通常存在着的是一种最简单的线性关系，这类数学模型可称谓线性的数据结构。

例 1-2 计算机和人对奕问题。计算机之所以能和人对奕是因为有人将对奕的策略事先已存入计算机。由于对奕的过程是在一定规则下随机进行的，所以，为使计算机能灵活对奕就必须对对奕过程中所有可能发生的情况以及相应的对策都考虑周全，并且，一个“好”的棋手在对奕时不仅要看棋盘当时的状态，还应能预测棋局发展的趋势，甚至最后结局。因此，在对奕问题中，计算机操作的对象是对奕过程中可能出现的棋盘状态——称为格局。例如图 1.2(a)所示为井字棋[①] 的一个格局，而格局之间的关系是由比赛规则决定的。通常，这个关系不是线性的，因为从一个棋盘格局可以派生出几个格局，例如从图 1.2(a)所示的格局可以派生出五个格局，如图 1.2(b)所示，而从每一个新的格局又可派生出四个可能出现的格局。因此，若将从对奕开始到结束的过程中所有可能出现的格局都画在一张图上，则可得到一棵倒长的“树”。“树根”是对奕开始之前的棋盘格局，而所有的“叶子”就是可能出现的结局，对奕的过程就是从树根沿树叉到某个叶子的过程。“树”可以是某些非数值计算问题的数学模型，它也是一种数据结构。

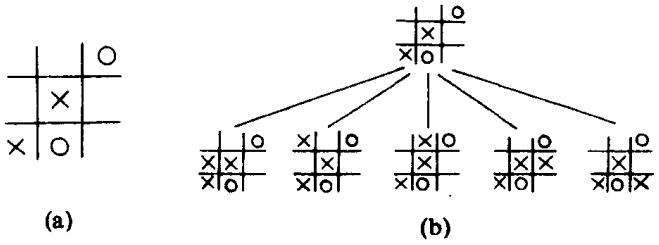

图 1.2 井字棋对奕“树”

(a) 棋盘格局示例；(b) 对奕树的局部。

例 1-3 多叉路口交通灯的管理问题。通常，在十字交叉的路口只需设红、绿两色的交通灯便可保持正常的交通秩序，而在多叉路口需设几种颜色的交通灯才能既使车辆相互之间不碰撞，又能达到车辆的最大流通呢？假设有一个如图 1.3(a)所示的五叉路口，其中 C 和 E 为单行道。在路口有 13 条可行的通路，其中有的可以同时通行，如 A→B 和 E→

① 井字棋由两人对奕。棋盘为 3×3 的方格，当一方的三个棋子占同一行、或同一列、或同一对角线时便为胜方。

C，而有的不能同时通行，如 E→B 和 A→D。那末，在路口应如何设置交通灯进行车辆的管理呢？

通常，这类交通、道路问题的数学模型是一种称谓“图”的数据结构。例如在此例的问题中，可以图中一个顶点表示一条通路，而通路之间互相矛盾的关系以两个顶点之间的连线表示。如在图 1.3(b)中，每个圆圈表示图 1.3(a)所示五叉路口上的一条通路，两个圆圈之间的连线表示这两个圆圈表示的两条通路不能同时通行。则设置交通灯的问题等价为对图的顶点的染色问题，要求对图上的每个顶点染一种颜色，并且要求有线相连的两个顶点不能具有相同颜色，而总的颜色种类应尽可能地少。图 1.3(b)所示为一种染色结果，圆圈中的数字表示交通灯的不同颜色，例如 3 号色灯亮时只有 D→A 和 D→B 两条路可通行。

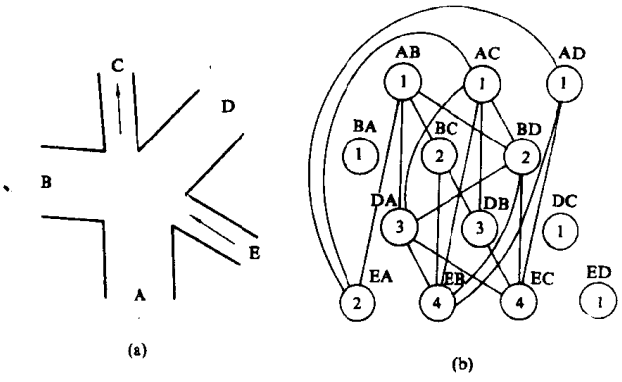

图 1.3　五叉路口交通管理示意图
(a) 五叉路口；(b) 表示通路的图。

综上三个例子可见，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树和图之类的数据结构。因此，简单说来，数据结构是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等等的学科。

1.2　基本概念和术语

在本节中，我们将对一些概念和术语赋以确定的含义，以便与读者取得“共同的语言”。这些概念和术语将在以后的章节中多次出现。

数据(data)　是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。它是计算机程序加工的“原料”。例如，一个利用数值分析方法解代数方程的程序，其处理对象是整数和实数；一个编译程序或文字处理程序的处理对象是字符串。因此，对计算机科学而言，数据的含义极为广泛，如图象、声音等都可以通过编码而归之于数据的范畴。

数据元素(data element)　是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。例如，例 1-2 中的“树”中的一个棋盘格局，例 1-3 中的“图”中的一个圆圈都被称为一个数据元素。有时，一个数据元素可由若干个**数据项**(data item)组成，例如，例 1-1 中一本书的书目信息为一个数据元素，而书目信息中的每一项(如书名、作者名等)为一个数据项。数据项是数据的不可分割的最小单位。

数据对象(data object)　是性质相同的数据元素的集合，是数据的一个子集。例如，整数数据对象是集合 N＝{0，±1，±2，…}，字母字符数据对象是集合 C＝{‘A’，‘B’，…，‘Z’}。

数据结构(data structure)　是相互之间存在一种或多种特定关系的数据元素的集合。这是本书对数据结构的一种简单解释。[①] 从上节三个例子可以看到，在任何问题中，数据元素都不是孤立存在的，而是在它们之间存在着某种关系，这种数据元素相互之间的关系称为**结构**(structure)。根据数据元素之间关系的不同特性，通常有下列四类基本结构：(1)**集合**　结构中的数据元素之间除了"同属于一个集合"的关系外，别无其它关系[②]；(2)**线性结构**　结构中的数据元素之间存在一个对一个的关系；(3)**树形结构**　结构中的元素之间存在一个对多个的关系；(4)**图状结构**或**网状结构**　结构中的元素之间存在多个对多个的关系。图 1.4 为上述四类基本结构的关系图。由于"集合"是元素之间关系极为松散的一种结构，因此也可用其它结构来表示它。

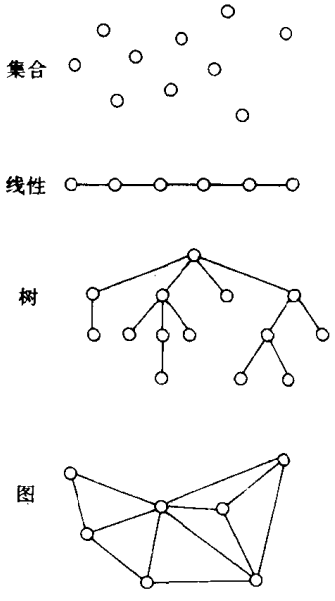

图 1.4　四类基本结构关系图

数据结构的形式定义为：数据结构是一个二元组

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

其中：D 是数据元素的有限集，S 是 D 上关系的有限集。下面举两个简单例子说明之。

例 1-4　在计算机科学中，复数可取如下定义：复数是一种数据结构

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

其中：C 是含两个实数的集合{c1,c2}；R={P}，而 P 是定义在集合 C 上的一种关系{⟨c1,c2⟩}，其中有序偶⟨c1,c2⟩表示 c1 是复数的实部，c2 是复数的虚部。

例 1-5　假设我们需要编制一个事务管理的程序，管理学校科学研究课题小组的各项事务，则首先要为程序的操作对象——课题小组设计一个数据结构。假设每个小组由一位教师，一至三名研究生及一至六名本科生组成，小组成员之间的关系是：教师指导研究生，而由每位研究生指导一至二名本科生。则可以如下定义数据结构：

$$\text{Group} = (P,R) \tag{1-3}$$

其中：$P=\{T,G_1,\cdots,G_n,S_{11}\cdots S_{nm}\}$[③] $_{1\leqslant n\leqslant 3,1\leqslant m\leqslant 2}$，

$R=\{R_1,R_2\}$

$R_1=\{\langle T,G_i\rangle \mid_{1\leqslant i\leqslant n,1\leqslant n\leqslant 3}\}$

$R_2=\{\langle G_i,S_{ij}\rangle \mid_{1\leqslant i\leqslant n,1\leqslant j\leqslant m,1\leqslant n\leqslant 3,1\leqslant m\leqslant 2}\}$

上述数据结构的定义仅是对操作对象的一种数学描述，换句话说，是从操作对象抽象出来的数学模型。结构定义中的"关系"描述的是数据元素之间的逻辑关系，因此又称为数

① 对于数据结构这个概念，至今尚未有一个被一致公认的定义，不同的人在使用这个词时所表达的意思有所不同。

② 这和数学中的集合概念是一致的。

③ T 表示导师，G 表示研究生，S 表示大学生。

据的**逻辑结构**。然而，讨论数据结构的目的是为了在计算机中实现对它的操作，因此还需研究如何在计算机中表示它。

数据结构在计算机中的表示（又称映象）称为数据的**物理结构**，又称**存储结构**。它包括数据元素的表示和关系的表示。在计算机中表示信息的最小单位是二进制数的一位，叫做**位**（bit）。在计算机中，我们可以用一个由若干位组合起来形成的一个位串表示一个数据元素（如用一个字长的位串表示一个整数，用八位二进制数表示一个字符等），通常称这个位串为**元素**[①]（element）或**结点**（node）。当数据元素由若干数据项组成时，位串中对应于各个数据项的子位串称为**数据域**（data field）。因此，元素或结点可看成是数据元素在计算机中的映象。

数据元素之间的关系在计算机中有两种不同的表示方法：**顺序映象**和**非顺序映象**，并由此得到两种不同的存储结构：**顺序存储结构**和**链式存储结构**。顺序映象的特点是借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系。例如，假设用两个字长的位串表示一个实数，则可以用地址相邻的四个字长的位串表示一个复数，如图 1.5(a)为表示复数z1＝3.0－2.3i和z2＝－0.7＋4.8i的顺序存储结构；非顺序映象的特点是借助指示元素存储地址的**指针**（pointer）表示数据元素之间的逻辑关系，如图 1.5(b)为表示复数 z 1 的链式存储结构，其中实部和虚部之间的关系用值为“0415”的指针来表示（0415 是虚部的存储地址）[②]。数据的逻辑结构和物理结构是密切相关的两个方面，以后读者会看到，任何一个算法的设计取决于选定的数据（逻辑）结构，而算法的实现依赖于采用的存储结构。

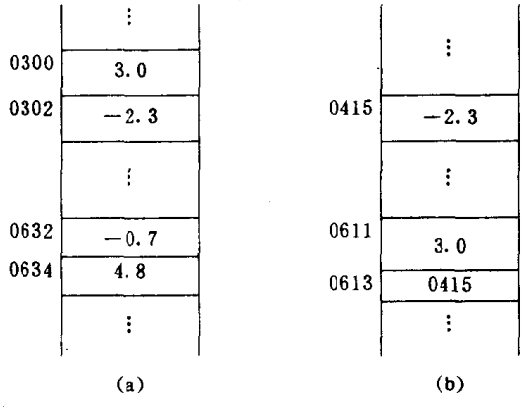

图 1.5　复数存储结构示意图

(a) 顺序存储结构；(b) 链式存储结构。

如何描述存储结构呢。虽则存储结构涉及数据元素及其关系在存储器中的物理位置，但由于本书是在高级程序语言的层次上讨论数据结构的操作，因此不能如上那样直接以

① 本书中有时也把数据元素简称为元素，读者应从上下文去理解分辨之。

② 在实际应用中，象复数这类极简单的结构不需要采用链式存储结构，在此仅为了简化讨论而作为假例引用之。

内存地址来描述存储结构，但我们可以借用高级程序语言中提供的“数据类型”来描述它，例如可以用所有高级程序语言中都有的“一维数组”类型来描述顺序存储结构，以 PASCAL 语言提供的“指针”来描述链式存储结构。假如我们把 PASCAL 语言看成是一个执行 PASCAL 指令和 PASCAL 数据类型的虚拟处理器，那末本书中讨论的存储结构是数据结构在 PASCAL 虚拟处理器中的表示，不妨称它为**虚拟存储结构**。

数据类型(Data type) 是和数据结构密切相关的一个概念，它最早出现在高级程序语言中，用以刻划(程序)操作对象的特性。在用高级程序语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。类型明显或隐含地规定了在程序执行期间变量或表达式所有可能取值的范围，以及在这些值上允许进行的操作。因此数据类型是一个值的集合和定义在这个值集上的一组操作的总称。例如 PASCAL 语言中的整数类型，其值集为区间[－maxint，maxint]上的整数(maxint 是依赖特定的计算机的最大整数)，定义在其上的一组操作为：加、减、乘、整除和取模等。

按“值”的不同特性，高级程序语言中的数据类型可分为两类：一类是非结构的**原子类型**。原子类型的值是不可分解的。如 PASCAL 语言中的标准类型(整型、实型、字符型和布尔型)、枚举类型、子界类型和指针类型；另一类是**结构类型**。结构类型的值是由若干成分按某种结构组成的，因此是可以分解的，并且它的成分可以是非结构的，也可以是结构的。例如数组的值由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，则结构类型可以看成由一种数据结构和定义在其上的一组操作组成。

实际上，在计算机中，数据类型的概念并非局限于高级语言中，每个处理器①(包括计算机硬件系统、操作系统、高级语言、数据库等)都提供了一组原子类型或结构类型。例如，一个计算机硬件系统通常含有“位”、“字节”、“字”等原子类型，它们的操作通过计算机设计的一套指令系统直接由电路系统完成，而高级程序语言提供的数据类型，其操作需通过编译器或解释器转化成低层即汇编语言或机器语言的数据类型来实现。引入“数据类型”的目的，从硬件的角度看，是作为解释计算机内存中信息含义的一种手段，而对使用数据类型的用户来说，实现了信息的隐蔽，即将一切用户不必了解的细节都封装在类型中。例如，用户在使用“整数”类型时，既不需要了解“整数”在计算机内部是如何表示的，也不需要知道其操作是如何实现的。如“两整数求和”，程序设计者注重的仅仅是其“数学上求和”的抽象特性，而不是其硬件的“位”操作如何进行。

抽象数据类型(Abstract Data Type 简称 ADT) 是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关，即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

抽象数据类型和数据类型实质上是一个概念。例如，各个计算机都拥有的“整数”类型是一个抽象数据类型，尽管它们在不同处理器上实现的方法可以不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此，“抽象”的意义在于数据类型的数学抽象特性。

---

① 在此指广义的处理器，包括计算机的硬件系统和软件系统。

另一方面，抽象数据类型的范畴更广，它不再局限于前述各处理器中已定义并实现的数据类型(也可称这类数据类型为固有数据类型)，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的复用率，在近代程序设计方法学中指出，一个软件系统的框架应建立在数据之上，而不是建立在操作之上(后者是传统的软件设计方法所为)。即在构成软件系统的每个相对独立的模块上，定义一组数据和施于这些数据上的一组操作，并在模块内部给出这些数据的表示及其操作的细节，而在模块外部使用的只是抽象的数据和抽象的操作。显然，所定义的数据类型的抽象层次越高，含有该抽象数据类型的软件模块的复用程度也就越高。

一个含抽象数据类型的软件模块通常应包含定义、表示和实现三个部分。

如前所述，抽象数据类型的定义由一个值域和定义在该值域上的一组操作组成。若按其值的不同特性，可细分为下列三种类型：

原子类型(atomic data type)。属原子类型的变量的值是不可分解的。这类抽象数据类型较少，因为一般情况下，已有的固有数据类型足以满足需求。但有时也有必要定义新的原子数据类型，例如数位为 100 的整数。

固定聚合类型(fixed-aggregate data type)。属该类型的变量，其值由确定数目的成分按某种结构组成。例如，后面将要定义的复数是由两个实数依确定的次序关系构成。

可变聚合类型(variable-aggregate data type)。和固定聚合类型相比较，构成可变聚合类型“值”的成分的数目不确定。例如，可定义一个“有序整数序列”的抽象数据类型，其中序列的长度是可变的。

显然，后两种类型可统称为结构类型。

抽象数据类型定义的操作中，除了和该类型相应的一组操作外，一般情况下还应有结构的创建和销毁。

抽象数据类型可通过固有数据类型来表示和实现，即利用处理器中已存在的数据类型来说明新的结构，利用已经实现的操作来进行新的操作。正如前面所提到的，由于本书是在高级程序语言的层次上进行讨论，因此对抽象数据类型也只是在这虚拟层上讨论它的实现。下面以复数为例说明之。

例 1-6 抽象数据类型复数的定义。

例 1-4 中式(1-2)的逻辑结构定义实际上是给定了复数的值域，为两个实数集的笛卡尔乘积。即

$$Z = R \times R = \{\langle c1, c2\rangle \mid c1 \in R, c2 \in R\} \tag{1-4}$$

其中 R 表示实数集，Z 表示复数集。

下面定义复数的六种操作：

1) **CREATE**(x,y,z) 生成一个复数。

对任何一对实数 x,y(x∈R,y∈R)，必可生成一个复数域中的复数 z=x+iy。

2) **ADD**(z1,z2,sum)复数求和。

对复数域 Z 中的任意两个复数 z1=x1+iy1 和 z2=x2+iy2，必可求得其和为 sum=(x1+x2)+i(y1+y2)。

3) **SUBTRACT**(z1,z2,difference)复数求差。

对复数域 Z 中的任意两个复数 z1=x1+iy1 和 z2=x2+iy2，必可求得其差为 difference=(x1−x2)+i(y1−y2)。

4) **MULTIPLY**(z1,z2,product)　复数求积。

对复数域 Z 中的任意两个复数 z1=x1+iy1 和 z2=x2+iy2，必可求得其积为 product=(x1·x2−y1·y2)+i(x1·y2+x2·y1)。

5) **GET_REALPART**(z)　取复数的实部。

对复数域 Z 中的任意一个复数 z=x+iy，必可求得其实部 x 且 x∈R。

6) **GET_IMAGPART**(z)　取复数的虚部。

对复数域 Z 中的任意一个复数 z=x+iy，必可求得其虚部 y 且 y∈R。

以上对复数结构的定义(包括组成复数值的成分的值域和成分间的关系，如式(1-2)所示)及其六种操作的定义构成了抽象数据类型"复数"的定义，或者说是它的规范说明。不论它在计算机内部如何实现，对使用它的外部用户来说，只需了解并严格遵循上述抽象数学特性即可。

实现抽象数据类型需借助于高级程序语言，但具体实现细节依赖于所用语言的功能，下面分三种情况讨论之。

第一种情况：在标准 PASCAL 等面向过程的语言中，用户可以自己定义数据类型。由此可以借助过程或函数、利用固有数据类型来表示和实现抽象数据类型。例如，借助标准 PASCAL 语言可如下表示例 1-6 中定义的抽象数据类型复数并实现它的运算。

```
TYPE cmptp=RECORD                    {复数类型}
                realpart : real;     {实部}
                imagpart : real      {虚部}
             END;
PROCEDURE create(x,y : real; VAR z : cmptp);
  {生成一个其实部为 x、虚部为 y 的复数 z}
BEGIN
    z.realpart := x;
    z.imagpart := y
END;  {create}
PROCEDURE add(z1,z2 : cmptp;VAR sum : cmptp);
  {求得和 sum=z1+z2=(x1+iy1)+(x2+iy2)=(x1+x2)+i(y1+y2)}
BEGIN
    sum.realpart := z1.realpart+z2.realpart;
    sum.imagpart := z1.imagpart+z2.imagpart
END ;  {add}
PROCEDURE subtract (z1,z2 : cmptp;  VAR difference : cmptp);
  {求得差 difference=z1−z2=(x1+iy1)−(x2+iy2)=(x1−x2)+i(y1−y2)}
BEGIN
        {略}
END ;  {subtract}
PROCEDURE multiply (z1,z2 : cmptp; VAR product : cmptp);
```

```
  {求得积 product=z1 · z2=(x1+iy1) · (x2+iy2)=(x1 · x2-y1 · y2)+i(x1 · y2+x2 · y1)}
BEGIN
        {略}
END ;  {multiply}
FUNCTION get_realpart(z : cmptp) : real;
  {求得复数 z=x+iy 的实部 x}
BEGIN
    get_realpart : =z. realpart;
END ;  {get_realpart}
FUNCTION  get_imagpart(z : cmptp) : real;
  {求得复数 z=x+iy 的虚部 y}
BEGIN
  get_imagpart : =z. imagpart
END ;  {get_imagpart)
```

凡属 cmptp 类型的变量均可调用上述过程或函数进行操作，而不需要访问变量的数据域，从而实现了信息的隐蔽和封装。例如，假设

```
VAR  c1,c2,p : cmptp;
```

则调用过程语句

```
multiply (c1,c2,p);
```

执行结果求得两复数 c1 和 c2 的乘积 p=c1 · c2。

由于标准 PASCAL 语言的程序结构框架是由严格规定次序的“段”(包括程序首部、标号说明、常量定义、类型定义、变量说明、过程或函数说明，语句部分)组成，因此，所有使用复数类型的外部用户必须将上述复数的类型说明和过程说明嵌入自己程序的适当位置(即上述类型说明、变量说明、过程说明和过程调用语句必须依次装入同一程序内进行编联)。可见这类语言利用抽象数据类型进行程序设计的基本方法是，限制不拥有某个数据结构的模块不能访问该数据结构，称之为数据结构受限访问。

第二种情况：在 Ada 和 Module-2 等语言中，提供了“包”(package)或“模块”(module)的结构。每个模块可含一个或多个抽象数据类型，它不仅可单独进行编译，而且为外部使用抽象数据类型提供方便。下面以 TURBO PASCAL 4.0 为例进行讨论。

TURBO PASCAL 4.0 中提供一个类似模块的结构——单元(unit)。一个单元是常数、数据类型、过程及函数的集合，单元的结构类似于 PASCAL 程序，但又有很大不同。单元通过过程或函数提供一系列的功能，但这些功能的说明和实体分别放在单元的接口部分和实现部分中。接口部分好比是模块的窗口，它的内容对任何使用本单元的程序(或其它单元)都是“可见”的，可被任何使用本单元的程序(或单元)引用；实现部分好比是黑匣子，它的内容对任何使用本单元的程序都是不可见的。例如，下述为抽象数据类型复数的单元模块。

```
UNIT complex;
INTERFACE  {单元的接口部分}
  TYPE cmptp=RECORD
              realpart,imagpart : real
            END;
```

```
  PROCEDURE create(x,y : real;VAR z : cmptp);
  PROCEDURE add(z1,z2 : cmptp; VAR sum : cmptp);
  PROCEDURE subtract(z1,z2 : cmptp; VAR difference : cmptp);
  PROCEDURE  multiply(z1,z2 : cmptp; VAR product : cmptp);
  FUNCTION get_realpart(z : cmptp) : real;
  FUNCTION get_imagpart(z : cmptp) : real;
IMPLEMENTATION  {实现部分}
  {这里是上述说明部分中过程或函数的实体及单元私有的常量类型和变量说明}
BEGIN
  {单元初始化编码,对本单元是空语句}
END.
```

单元的设置突破了第一类情况中所述对数据结构访问权限的限定。当上述单元被编译[①]好之后,任何一个程序或单元均可通过 uses 子句引用单元接口部分中说明的常量、类型、变量、过程和函数。例如下述是一个引用复数单元的程序例子。

```
PROGRAM example;
  USES complex;
  VAR rp,ip : real; c1,c2,p : cmptp;
  BEGIN
    read(rp,ip);  create(rp,ip,c1);
    read(rp,ip);  create(rp,ip,c2);
    multiply(c1,c2,p);
    writeln(get_realpart(p),'+i',get_imagpart(p))
  END.
```

在本书以下各章的讨论中,有时将借用这种"UNIT"的模块结构来说明抽象数据类型的表示和实现。

第三种情况:在面向对象的程序设计(object_Oriented Programming 简称 OOP)语言中,借助对象(Object)描述抽象数据类型。

从前面对数据类型的讨论可看到,"类型"的概念与"操作"是密切相关的,同一种数据结构和不同的操作组将构成不同的数据类型。在面向对象的程序设计语言中,结构说明和过程说明被统一在一个整体——**对象**之中,其中数据结构的定义为对象的属性域,过程或函数定义在对象中称之为**方法**(methed),是对象的性能描述(即能进行何种操作)。例如,以下是一个复数对象的简单例子。

```
  TYPE
    complex=OBJECT
      realpart,imagpart : real;
      PROCEDURE creat(x,y : real);{创建一个复数}
      PROCEDURE clear;  {销毁已存在的结构}
```

① 详细情况请参阅 TURBO PASCAL 4.0 的使用手册。

```
    FUNCTION getrealpart : real;
    FUNCTION getimagpart : real
  END;
```

从上述例子可见，对象说明和类型说明不同，其特点是将抽象数据类型的全部信息（结构、成分的属性及过程或函数说明）都封装在其自身之中。一旦定义了一个对象，就可以使用对象的名字来说明变量，如在变量说明

```
VAR   cmx : complex;
```

之后，在程序中就可出现如下语句：

```
WITH cmx DO BEGIN
  creat(x,y);
  writeln(getrealpart,getimagpart);
  clear
END;
```

值得注意的是，面向对象程序设计语言的特点不仅是**封装**(encapsulation)，还有**继承**(inheritance)和**多型**(polymorphism)等。因此，用它来描述“复数”这类简单的抽象数据类型显然是大材小用了。

多形数据类型(polymorphic data type)　是指其值的成分不确定的数据类型。例 1-6 中定义的抽象数据类型复数和整数类型类似，有一个确定的值域，即每个“复数”是由两个“实数”依一定次序构成。在实际问题中，我们经常会碰到一些结构相同而值的成分不同的数据结构。例如：“有序序列”是一种很有用的数据结构，序列中的元素可以是整数、或字符、字符串甚至更复杂的复合成份。然而，不论其元素是简单类型，还是复杂类型，常用的操作相同。例如：在序列中检索某个元素；插入或删除某个元素等。如果我们将它设定为某种特定元素类型的有序序列类型，则需要定义多个操作类似的数据类型，如：“整数有序序列”、“字符串的有序序列”和“复数有序序列”等。为了避免重复，提高软件复用率，我们需定义“元素的有序序列”这样一个抽象数据类型。由于在类型定义中不考虑元素的属性，仅从元素之间的结构关系抽象而得，故称为多形数据类型。显然，需借助面向对象的程序设计语言实现之。本书中讨论的各种数据结构大多属多形数据结构，限于不增加课程的难度，讨论中均不定义成多形数据类型。

从以上对数据类型的讨论中可见，与数据结构密切相关的是定义在数据结构上的一组操作。操作的种类和数目不同，即使逻辑结构相同，这个数据结构的用途也会大为不同（最典型的例子是第三章所讨论的栈和队列）。

操作的种类是没有限制的，可以根据需要而定义。基本的操作主要有以下几种：

1) **插入**　在数据结构中的指定位置上增添新的数据元素；

2) **删除**　删去数据结构中某个指定的数据元素；

3) **更新**　改变数据结构中某个数据元素的值，在概念上等价于删除和插入操作的组合；

4) **查找**　在数据结构中寻找满足某个特定要求的数据元素(的位置和值)。

5) **排序**　(在线性结构中)重新安排数据元素之间的逻辑顺序关系，使之按值由小到大或由大到小的次序排列。

从操作的特性来分，所有的操作可以归结为两类：一类是**加工型操作**(constructor)，操作改变了(操作之前的)结构的值；另一类是**引用型操作**(selector)，即此操作不改变结构的值，只是查询或求得结构的值。不难看出，前述五种操作中除"查找"为引用型操作外，其余都是加工型操作。

算法(algorithm)　是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作；此外，一个算法还具有下列五个重要特性：

1) **有穷性**　一个算法必须总是(对任何合法的输入值)在执行有穷步之后结束，且每一步都可在有穷时间① 内完成；

2) **确定性**　算法中每一条指令必须有确切的含义，读者理解时不会产生二义性。并且，在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。

3) **可行性**　一个算法是能行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

4) **输入**　一个算法有零个或多个的输入，这些输入取自于特定的对象的集合。

5) **输出**　一个算法有一个或多个的输出。这些输出是同输入有某个特定关系的量。

在 1.4 节中我们将进一步讨论算法的描述和度量。

1.3　数据结构的发展简史及它在计算机科学中所处的地位

《数据结构》作为一门独立的课程在国外是从 1968 年才开始设立的。在这之前，它的某些内容曾在其它课程，如表处理语言中有所阐述。1968 年在美国一些大学的计算机系的教学计划中，虽然把《数据结构》规定为一门课程，但对课程的范围仍没有作明确规定。当时，数据结构几乎和图论，特别是和表、树的理论为同义语。随后，数据结构这个概念被扩充到包括网络、集合代数论、格、关系等方面，从而变成了现在称之为《离散结构》的内容。然而，由于数据必须在计算机中进行处理，因此，不仅考虑数据本身的数学性质，而且还必须考虑数据的存储结构，这就进一步扩大了数据结构的内容。近年来，随着数据库系统的不断发展，在数据结构课程中又增加了文件管理(特别是大型文件的组织等)的内容。

1968 年美国唐·欧·克努特教授开创了数据结构的最初体系，他所著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作，从 60 年代末到 70 年代初，出现了大型程序，软件也相对独立，结构程序设计成为程序设计方法学的主要内容，人们就越来越重视数据结构，认为程序设计的实质是对确定的问题选择一种好的结构，加上 设计一种好的算法。从 70 年代中期到 80 年代初，各种版本的数据结构著作就相继出现。

目前在我国，《数据结构》也已经不仅仅是计算机专业的教学计划中的核心课程之一，而且是其它非计算机专业的主要选修课程之一。

---

① 在此，有穷的概念不是纯数学的，而且实际上是合理的，可接受的。