面向21世纪计算机专业规划教材

# 数据结构与算法设计

周海英 马巧梅 靳雁霞 编著

SHUJU JIEGOU YU SUANFA SHEJI

国防工业出版社
National Defense Industry Press

面向21世纪计算机专业规划教材

# 数据结构与算法设计

周海英　马巧梅　靳雁霞　编著

国防工业出版社
·北京·

# 内容简介

本书主要介绍了数据结构的基本概念和基本算法。全书共分11章。前6章主要介绍了线性表、栈和队列、串、递归、数组特殊矩阵和广义表，后5章主要介绍了树、图、查找、排序和文件。

本书内容详细，基本原理与算法实现相互结合并配套了大量典型例题，便于初学者掌握重要的概念、原理和算法设计方法，也方便读者复习该门课程的重要知识点。

本书可作为高等院校计算机及相关专业本科生数据结构课程的教材，也可作为计算机工程技术人员学习的参考书。

**图书在版编目(CIP)数据**

数据结构与算法设计/周海英，马巧梅，靳雁霞编著.
北京：国防工业出版社，2007.8
面向21世纪计算机专业规划教材
ISBN 978-7-118-05254-1

Ⅰ.数... Ⅱ.①周...②马...③靳... Ⅲ.①数据结构－高等学校－教材②电子计算机－算法设计－高等学校－教材 Ⅳ.TP311.12 TP301.6

中国版本图书馆CIP数据核字(2007)第098507号

※

国防工业出版社出版发行
(北京市海淀区紫竹院南路23号 邮政编码100044)
新艺印刷厂印刷
新华书店经售
*
**开本** 787×1092 1/16 **印张** 21 **字数** 528千字
2007年8月第1版第1次印刷 **印数** 1—4000册 **定价** 39.00元

**(本书如有印装错误，我社负责调换)**

国防书店：(010)68428422 发行邮购：(010)68414474
发行传真：(010)68411535 发行业务：(010)68472764

# 前　言

数据结构课程作为国内计算机及信息管理等相关专业的重要专业基础课已有二十多年的教学实践了。该门课程会给其他计算机专业课程的学习打下重要的知识基础，它已成为计算机类本专科学生的核心课程。从国内外的教学情况可以看出，数据结构课程的基本内容体系已渐趋成熟，其中的一些内容吸收了离散数学中的部分理论，为计算机专业课程的学习和软件开发提供了重要的理论和方法基础。本书的编写根据国内该课程的教学情况以及研究生入学考试中数据结构的重要知识点，参考了大量的相关教材后编写完成，既可以作为本专科数据结构课程的教材和参考书，又可作为参加研究生考试的学生的辅导教材。

本书在编写中以理论和实践相结合，根据作者多年的教学实践总结出的经验，在介绍数据结构算法和一般理论的同时给出了大量的典型例题和习题，便于学生在掌握理论方法的同时能熟练把握课程重要的知识点和方法，并通过对例题的消化理解以及习题练习迅速掌握重要的解题方法和算法设计技巧，为提高学习的效率和效果提供良好的帮助。

本书在第 1 章到第 4 章中主要介绍了数据结构的基本概念、线性表以及栈和队列等线性结构的基本知识，这些内容是数据结构的重要基础，在内容上涉及大量的概念和算法。本书在编写时给出了大量的例题，其中一些就是定义和基本概念方面的问答题，通过这些内容使同学们了解数据结构研究的基本内容和重要方法，为后面章节的学习打好基础。第 5 章介绍了递归，内容主要与算法分析与设计课程有联系，在讲授时可以根据教学要求进行选择。第 6 章介绍了数组、特殊矩阵和广义表方面的内容，与前面章节的内容有关联，但在内容的逻辑性上又有一定的独立性。第 7 章、第 8 章主要介绍树和图两种常用的非线性结构，它们是数据结构中的重点内容之一，对解决实际问题提供了重要的方法论基础，本书安排了大量篇幅给予介绍。第 9 章和第 10 章介绍的排序和查找是实际应用中最常见的问题，对于算法设计和编程有重要的帮助，也是数据结构中的另一个重点内容之一。第 11 章介绍文件，这章的内容可以根据课时情况作为选讲内容。

全书共分 11 章。第 1 章、第 7 章、第 10 章、第 11 章由周海英编写，第 2 章、第 5 章、第 6 章、第 8 章由马巧梅编写，第 3 章、第 4 章、第 9 章由靳雁霞编写。

本书在编写中参考了国内外许多文献资料（详见参考文献），在此，向以上文献的作者表示感谢。

本书的初稿完成后，山西大学的任瑞征老师认真审阅了书稿并提出了许多宝贵的意见；在编写过程中，得到了张清爽等同志的大力支持，在此表示感谢。

由于作者知识有限，本书中可能存在不少缺点和错误，恳请读者给予批评指正。

编著者

2007 年 5 月

# 目　录

# 第1章 绪 论

自从第一台电子计算机问世以来，计算机科学得到了飞速的发展，与此同时，计算机的应用领域也从最初的科学计算逐步发展到更高级的阶段，如图像处理、语音识别、机器翻译、人工智能等多个领域。现在计算机处理的对象不仅是简单的数值或字符，而是带有不同结构的各种数据。因此，要设计一个好的软件，除了要掌握一定的计算机程序设计语言之外，还必须研究各类数据的特性以及数据之间存在的关系。这是因为计算机要加工处理数据，必须能够将数据输入到计算机中，并能够以恰当的方式在计算机中表示并存储起来，还要便于根据需要对数据进行加工和处理，因而当各种数据输入计算机之前，必须先按某种数据的组织形式将数据组织好，然后还要考虑以什么样的方式进行存储，这种组织形式和存储方式直接关系到程序对数据的处理能力和处理效率，并影响到问题的解决。

综上所述，可以这样理解，计算机在解决一个问题时，先将问题中的有关对象抽取出来形成数据，并将这些数据组织在一起。为了要合理组织这些数据，就要先找到各个数据之间的逻辑关系，即数据的逻辑结构，从而选择一种合理的组织方式。其次，还要考虑计算机如何存储这些组织好的数据，即数据的物理结构或存储结构。因此，数据结构这门课程就是要解决两个主要的问题：数据的逻辑结构和数据的存储结构。

## 1.1 什么是数据结构

一般来说，当用计算机解决一个具体问题时，大致都需要经过下面几个步骤：首先要从具体问题中抽象出一个适当的数学模型(或数学公式)，然后设计一个描述此数学模型的算法，最后利用合适的程序设计语言来编写程序，进行测试、调整，直至最终得到满意的解答。抽象数学模型的过程实质上是分析问题，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的过程。事实上，有些问题的求解过程可以通过一定的方程进行一定的运算来获取。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却无法用数学方程加以描述。下面请看几个例子。

**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该书的时候；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，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：有按书名编排的，有按作者名编排的，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、仓库账目管理等。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。这类数学模型可称为线性的数据结构。

| | | | | |
|---|---|---|---|---|
| 001 | 高等数学 | 樊映川 | S01 | …… |
| 002 | 大学物理 | 王微微 | W01 | …… |
| 003 | 化学工程 | 郝兵 | H01 | …… |
| 004 | 计算机网络 | 万树蜂 | J01 | …… |
| …… | …… | …… | …… | …… |

| | |
|---|---|
| 高等数学 | 001, …… |
| 大学物理 | 002, …… |
| 计算机网络 | 004, …… |
| …… | …… |

| | |
|---|---|
| 樊映川 | 001, …… |
| …… | …… |

| | |
|---|---|
| W | 002, …… |
| S | 001, …… |
| J | 004, …… |
| …… | …… |

图 1-1　图书目录文件示例

**例 1-2**　酒店管理系统中的客房分配问题。

在酒店的客房房态管理过程中，希望同类房中各间客房的出租机会基本均等，以保证维持一个平均的磨损率。为此，分配客房采用的算法应该是“先退的房先被启用”。相应地，所有“空”的同类客房的管理模型应该是一个“队列”，即酒店前台每次接待客人入住时，从“队头”分配客房；当客人结账离开时，应将退掉的空客房排在“队尾”。由于“排队”是日常生活中经常需要的一种行为，因此队列也是这样一类活动的模拟程序中经常用的一种数学模型。

**例1-3**　铺设煤气管道问题。

假设要在某个城市的 n 个居民区之间铺设煤气管道，则在这 n 个居民之间只要铺设 n-1 条管道即可。假设任意两个居民区之间都可以架设管道，但由于地理环境的不同，所需经费也不同，则采用什么样的施工方案能使总投资尽可能少。这个问题即为“求图的最小生成树”的问题。其数学模型如图 1-2 所示。图中“顶点”表示居民区，顶点之间的连线及其上的数值表示可以架设的管道及所需经费。求解的算法为：在可能架设的 m 条管道中选取 n-1 条，既能连通 n 个居民区，又使总投资达到“最小”。

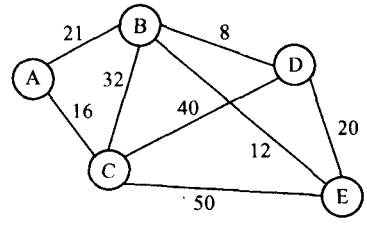


图 1-2　管道铺设问题

**例 1-4**　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，甚至最后的结局。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——格局，例如井字棋的一个格局，如图 1-3(a)所示，

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。图 1-3(a)所示的格局可以派生出五个格局，如图 1-3(b)所示，而从每一个新的格局又可派生出 n 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，也是一种数据结构。

综合上面几个例子可以看出，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关系和操作等的学科。

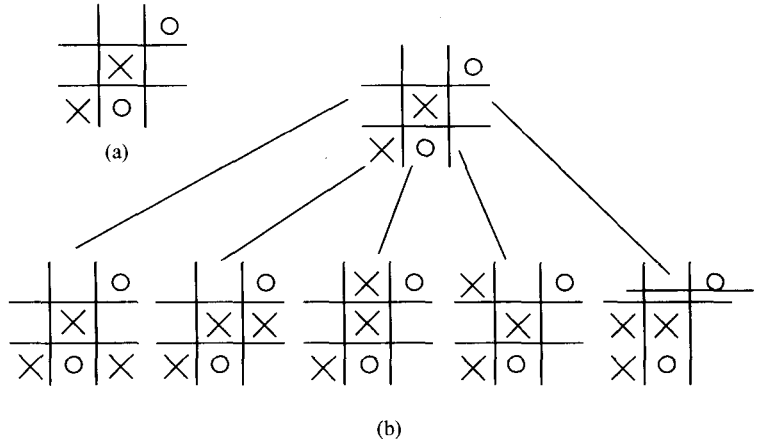


图 1-3　井字棋博弈树

## 1.2　基本概念和术语

在本节中，为了与读者就某些概念取得“共识”，将对一些概念和术语赋以确定的含义。这些概念和术语将在以后的章节中多次出现。

数据(data)是信息的载体，是对客观事物的符号表示。它能够被计算机所识别、存储和加工处理，是计算机程序加工的“原料”。数据是指所有能输入计算机，并被计算机程序处理的符号的总称。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，如图像、声音等都可以通过编码而归之于数据的范畴。

信息(information)是经过计算机加工处理的带有一定意义的结果，如方程式的解、遥感图像、视频信号等。

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 1-4 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，…}，大写字母字符数据对象是集合 C={'A'，'B'，…，'Z'}。

简单地说，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。通过上节的四个例子我们可以看到，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，别无其他关系。

(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：结构中的数据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。

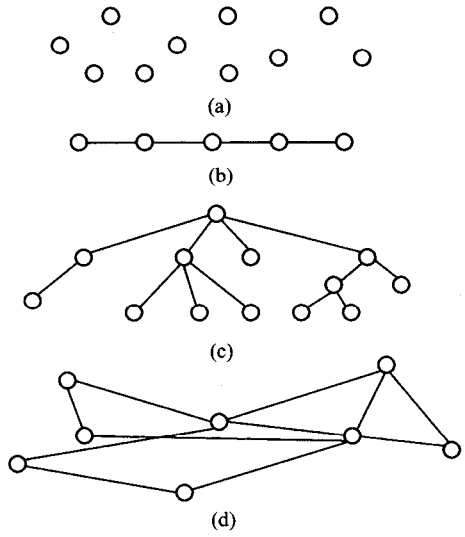


图 1-4　四种基本数据结构关系图

(a) 集合结构；(b) 线性结构；

(c) 树形结构；(d) 图状结构。

上述四类基本结构的关系如图 1-4 所示。由于"集合"是元素之间关系极为松散的一种结构，因此也可用其他结构来表示。

数据结构的形式定义为：

数据结构是一个二元组　　Data-Structure= (D，S)　　(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 1-5**　在计算机科学中，复数是一种数据结构：

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

其中：C 是含有两个实数的集合{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 1-6**　假设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 1 名教师、1 名～3 名研究生及 1 名～6 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 1 名～2 名本科生，则可以如下定义数据结构：

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

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

$R=\{R_1,R_2\}$

$R1=\{<T,G_1>|_{1\leqslant i<n,1\leqslant n\leqslant 3}\}$

$R2=\{<Gi,Sij>|_{1\leqslant i<n,1\leqslant j\leqslant m,1\leqslant n\leqslant 3,1\leqslant m\leqslant 2}\}$

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，它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。

存储结构(又称映像)是数据结构在计算机中的表示，也称为数据的物理结构。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位(bit)。计算机中，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 8 位二进制数表示一个字符等)，通常称这个位串为元素(element)或节点(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，元素或节点可看成是数据元素在计算机中的映像。

数据元素之间的关系在计算机中有两种不同的表示方法：顺序映像和非顺序映像，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。顺序映像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)所示为表示复数。Zl=3.0-3.2i 和 Z2=8.9-1.6i 的顺序存储结构。

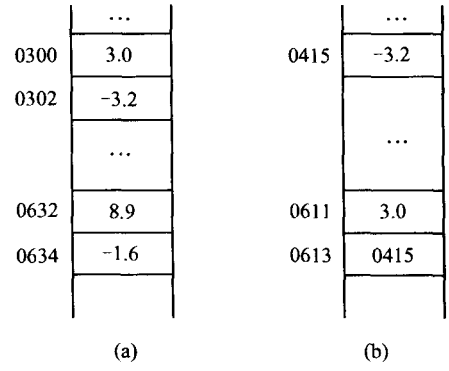


图 1-5　复数存储结构示意图

(a) 顺序存储结构；(b) 链式存储结构。

非顺序映像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)所示为表示复数 zl 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)。数据的逻辑结构和物理结构是密切相关的两个方面。在后面的章节读者会看到，任何一个算法的设计取决于所选定的数据(逻辑)结构，而算法的实现依赖于采用的存储结构。

那么，在数据结构确立之后如何描述存储结构呢?虽然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，因此不能如上所谈的那样直接以内存地址来描述存储结构。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 C 语言提供的“指针”来描述链式存储结构。假如把 C 语言看成是一个执行 C 指令和 C 数据类型的虚拟处理器，那么本书中讨论的存储结构是数据结构在 C 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)是与数据结构密切相关的一个概念，用以刻画(程序)操作对象的特性。它最早出现在高级程序语言中。在用高级程序语言编写的程序中，每个变量、常量或表达都有一个它所属的、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，以及在这些取值上所允许进行的操作。因此数据类型是一个值的集合和定义在这个值集合上的一组操作的总称。例如 C 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，另一类是结构类型。原子类型的值是不可分解的，如 C 语言中的标准类型(整型、实型、字符型、布尔型和指针类型)。结构类型的值是由若干成分按某种结构组成的，因此，它是可以分解的。它的成分既可以是非结构的，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(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级语言，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数据类型中。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，ADT)是指一个数学模型以及定义在该模型上的一组操作。ADT 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与实现无关，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。

ADT 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 ADT。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，在用户看来却是相同的。因此，“抽象”的意义在于数据类型的数学抽象特性。另一方面，ADT 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，近代程序设计方法学指出，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 ADT 的软件模块的复用程度也就越高。

一个含ADT的软件模块通常应包含定义、表示和实现三个部分。

ADT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ADT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的。

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ADT，其中序列的长度是可变的。

显然，我们可以把后两种类型统称为结构类型。在ADT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。

ADT可通过固有数据类型来表示和实现，即利用处理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ADT也只是在虚拟层上讨论它的实现。下面以复数为例说明。

**例1-7** ADT复数的定义。

例1-5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儿积，即

$$Z=R\times R=\{<c1,\ c2>|c1\in R,\ c2\in R\} \tag{1-4}$$

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(xffR，yffR)，必可生成一个复数域中的复数z=x+iy。

(2) ADD(zl，z2，sum)复数求和。

对复数域Z中的任意两个复数zl=xl+iyl和z2=x2+iy2，必可求得其和为

Sum=(xl+x2)+i(yl+y2)

(3) SUBTRACT(zl，z2，difference)复数求差。

对复数域Z中的任意两个复数z1=xl+iyl和z2=x2+iy2，必可求得其差为

difference=(xl−x2)+1(yl−y2)

(4) MULTIPLY(z1，z2，product)数求积。

对复数域z中的任意两个复数zl=xl+iyl和z2=x2+iy2，必可求得其积为

product=(xl • x2−y1 • y2)+i(xl • 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-IMAGPRT(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ADT“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ADT需要借助于高级程序语言，但具体实现细节则依赖于所用语言的功能。下面分两种情况讨论之。

第一种情况：在标准C语言等高级语言中，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ADT。例如，借助标准C语言可如下表示例1-6中定义的ADT复数，并实现它的运算。

```
Typedef Struct
{
int   realpart;      /*实部*/
int   imagpart;      /*虚部*/
}real_number
Real_number r_n;
```

**算法 1.1：**

```
Real_number *float create(int x,y;real_number z)
/*生成一个实部为x、虚部为y的复数z*/
{
z.realpart=x;
z.imagpart=Y;
return z;
};      /*create*/
```

**算法 1.2：**

```
add(real_number z1,z2;VAR real_number sum)
      /*求和sum=z1+z2=(x1+iy1)+(x2+iy2)=(x1+x2)+i(y1+y2)*/
        {Sum.realpart=z1.realpart+z2.realpart
        Sum.imagpart=z1.imagpart+z2.imagpart     };   /*add*/
```

**算法 1.3：**

```
subtract(real_number z1,z2;VAR real_number difference)
      /*求差difference=z1-z2=(x1+iy1)-(x2+iy2)=(x1-x2)+i(y1-y2)*/
{
Difference=(z1.x-z2.x)+i(z1.y-z2.y);
};      /*subtract*/
```

**算法 1.4：**

```
multiply(real_number z1，z2;VAR real number product)
      /*求积product=z1*z2=(x1+iy1)*(x2+iy2)=(x1*x2-y1*y2)+i(x1*y2+x2*y1)*/
{
   product=(z1.x*z2.x-z1.y*z2.y2)+i(z1.x*z2.y+z2.x*z1.y);
);/*multiply*/
```

**算法 1.5：**

```
float get_realpart(real_number z)
/*求复数z=x+iy的实部x*/
Get-realpart=z.realpart;
);/*get-realpart*/
```

**算法 1.6：**

```
float get_imagpart(real-number z)
/*求复数z=x+iy的虚部y*/
{
Get-imagpart=z.imagpart;
};/*get-imagpart*/
```

凡属real_number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。

第二种情况：在面向对象的程序设计(Objet-Oriented Programming，OOP)语言中，借助对象(object)描述ADT。

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OOP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(method)，是对象的性能描述(即能进行何种操作)。

值得注意的是，OOP语言的特点不仅是封装(encapsulation)，还有继承(in-heritance)和多型(polymorphism)等。因此，用它来描述“复数”这类简单的ADT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ADT复数与整数类型类似，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，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、字符、字符串，甚至是更复杂的复合成分。然而，不论其元素是简单类型，还是复杂类型，常用的操作都相同，例如在序列中检索某个元素和插入或删除某个元素等。如果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ADT。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3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(位置和值)。

(5) 排序：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)，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：一个算法是能行得通的，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输出。这些输出是同输入有某个特定关系的量。

在1.4节中将进一步讨论算法的描述和度量。

## 1.3 数据结构的发展及其重要地位

“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他课程，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计划中，虽然“数据结构”规定为一门课程，但对课程的范围仍没作明确规定。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，因此不仅要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·克努力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20世纪60年代末到20世纪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20世纪70年代中期到20世纪80年代初，各种版本的数据结构著作相继出现。

目前，“数据结构”在我国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，而且也是其他非计算机专业的主要选修课程之一。

“数据结构”在计算机科学中是一门综合性的专业基础课。数据结构的研究不仅涉及到计算机硬件(特别是编码理论、存储装置和存取方法等)的研究范围，而且和计算机软件的研究有着更密切的关系。无论是编译程序还是操作系统，都涉及到数据元素在存储器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以使查找和存取数据元素更为方便。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。在计算机科学中，数据结构不仅是一般程序设计(特别是非数值计算的程序设计)的基础，而且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。

值得注意的是，数据结构的发展并未停止。一方面，面向各专门领域中特殊问题的数据结构得到研究和发展，如多维图形数据结构等；另一方面，从ADT的观点来讨论数据结构，已成为一种新的趋势，越来越被人们所重视。

## 1.4 算法的描述和算法分析

### 1.4.1 算法的描述

算法需要用一种语言来描述，同时，算法可有各种描述方法以满足不同的需求。例如，一个需在计算机上运行的程序(程序也是算法)必须是严格按照语法规定用机器语言或汇编语言或高级程序语言编写的，而一个为了阅读和交流的算法可以用伪码语言或框图等其他形式来描述。由于本书中讨论的算法主要是为读者阅读，且能容易地转换成用高级语言书写的程序，因此采用介于伪码语言和程序之间的一种形式，即类C语言作为描述工具，有时也用伪码语言描述一些只含抽象操作的抽象算法。

本书中采用的类C语言是对标准C语言的一种简化及简单扩充，在此仅作简单说明。

(1) 所有的算法都以如下所示的过程或函数的形式表示。

```
函数类型 函数名(函数参数表)
{
/*算法说明*/
语句序列
}
```