数据结构实用教程

（C/C++描述）

徐孝凯 编著

清华大学出版社
http://www.tup.tsinghua.edu.cn

数据结构实用教程
(C/C++描述)

徐孝凯　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书是为全国高等院校计算机专业及相关专业开设数据结构课程而精心组织和编著的一本实用教材。本书从软件开发的实际需要出发,按照结构化和面向对象的程序设计思想,深入介绍了计算机处理的对象——数据的各种逻辑结构、存储结构以及进行查找、插入、删除、排序等运算的算法;对于每一种算法都利用了当前最流行和实用的 C/C++ 语言进行了具体实现,并全部上机通过;对于重点和难点内容,通过循序渐进的分析并结合事例加以说明,使得读者容易理解、掌握和运用。本书在内容安排上前后一致,连贯有序、层次分明、便于自学。另外,本书具有丰富的练习题,并配有习题参考解答一书同时出版。

书　　名:数据结构实用教程(C/C++ 描述)
作　　者:徐孝凯
出 版 者:清华大学出版社 (北京清华大学校内,邮政编码:100084)
　　http://www.tup.tsinghua.edu.cn
印 刷 者:清华大学印刷厂
发 行 者:新华书店总店北京科技发行所
开　　本:787×1092 1/16　印张:20.75　字数:429 千字
版　　次:1999 年 12 月第 1 版　2000 年 7 月第 2 次印刷
书　　号:ISBN 7-302-01950-9/TP·2199
印　　数:8001 ~ 14000
定　　价:29.00 元

前　言

数据结构是普通高校计算机专业和信息管理专业一门必修的核心课程。它的主要任务是讨论现实世界中数据(即事物的抽象描述)的各种逻辑结构,在计算机中的存储结构,以及进行各种非数值运算的算法。目的是使学生掌握数据组织、存储和处理的常用方法,为以后进行软件开发和学习后续专业课程打下基础。

数据的逻辑结构分为线性结构、树(层次)结构和图形结构三种。数据的存储结构是顺序(数组)结构、链接结构、索引结构和散列结构中的一种或多种的组合。对数据进行的非数值运算主要包括插入、删除、查找、排序、输入和输出等。需要特别指出:数据的存储结构既适用于内存,也适用于外存,不仅要学会对内存数据操作的算法,而且要学会对外存数据(文件)操作的算法,这样才能够解决实际软件开发的问题,达到学以致用的目的。

在已经出版的众多数据结构教材中,对每一种数据结构类型进行相应运算的算法描述通常是粗略的,离真正用一种计算机语言上机实现还有相当的距离,特别在外存文件的操作上更是如此。本套教材在这方面做了彻底的改变,所给出的每一算法都利用 C/C++ 语言进行了具体实现,算法的正确性和有效性得到了实际的检验,这样就突出了实用性,使教材更便于教学,特别是自学,克服了以往同类教材只重视理论而轻视具体实现的缺陷。

本套教材包含两本,一本为主教材《数据结构实用教程》,另一本为辅助教材《数据结构实用教程习题参考解答》。主教材共分为八章,依次为绪论、线性表、稀疏矩阵和广义表、栈和队列、树、图、查找、排序等。在第一章绪论中,第二节为算法描述,对于 C++ 语言比较熟悉的教学班和读者,此内容可作为自学阅读内容。主教材没有专门给出文件一章,这是因为,可把文件(这里只讨论磁盘文件)看做在外存上存储的数组,通过文件流操作函数既可顺序存取文件内容,也可随机存取文件中任何位置上的信息块,如何使用文件已经在 C/C++ 语言中学习过,在数据结构课程中只是应用问题,所以不应单列一章介绍。辅助教材给出了主教材中每章习题的大部分参考解答,最后增加了两道综合题,它们是针对不同文件的插入、删除和查找操作的,目的是培养学生对所学知识的实际应用能力。

本人一直从事数据结构的教学和研究工作,编著过多本教材,清华大学出版社已经出版的《数据结构简明教程》一书就是其代表作。现在这套教材是本人多年教学经验的结晶,是对以往教材的继承、丰富和发展,希望能够对数据结构课程的教学起到良好的作用,使读者能够得到满意的收获。

本套教材是根据计算机专业和信息管理专业本科培养目标对数据结构课程教学的要求编写的。由于内容叙述细致,算法描述具体,便于自学,所以删除部分内容后可作为相应专科学生的教材,具体如何删减,应由办学单位和任课教师定夺。

学习本套教材应具有 C 语言或 C++ 语言的基础。若只学习过 C 语言,则应在学习本课

程的过程中补充 C++ 语言的输入输出操作、文件流操作、运算符重载等有关内容。

承蒙北京大学计算机系许卓群教授和北京石油大学计算机系陈明教授认真审阅了本套教材的全部书稿，提出了宝贵的意见，在此谨向他们表示衷心感谢。

尽管本人做了很大的努力，但由于水平有限，加之时间仓促，错误和不足之处在所难免，敬请授课教师和广大读者批评指正，本人不胜感激，以便在教材重印或重版时能够得到不断地修改和完善，从而以更好地教材质量服务于读者。

电子邮件地址：xuxk@crtvu.edu.cn

联系电话：010-66052930

徐孝凯

1999 年 10 月

第一章 绪 论

自 1946 年美国第一台电子计算机问世以来，计算机科学和软硬件技术得到了飞速的发展，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人类活动的各个领域。现在，计算机处理的对象不仅是简单的数值和字符，而且还包括图形、图像、声音等多媒体数据，其数据的结构也愈加复杂。因此，要开发出一种性能良好的软件，不仅要根据实际需要掌握至少一种适合的计算机高级语言或软件开发工具，更重要的是研究数据的不同结构和组织方法以及进行数据处理的不同算法，通过分析和比较选择出较好的设计方案才行。后者正是数据结构这门学科所要解决的问题。

1.1 基本术语

这一节，我们将对书中一些常用的名词和术语赋以确定的含义。

数据(Data)是人们利用文字符号、数字符号以及其他规定的符号对现实世界的事物及其活动所做的抽象描述。因此，一个文档、记录、数组、句子、单词、算式、符号等都统称为数据。在计算机领域，人们把能够被计算机加工的对象，或者说能够被计算机输入、存储、处理和输出的一切信息都叫做数据。

数据元素(Data Element)简称**元素**，它是一个数据整体中相对独立的单位。如对于一个文件来说，每个记录就是它的数据元素；对于一个字符串来说，每个字符就是它的数据元素；对于一个数组来说，每一个成分就是它的数据元素。数据和数据元素是相对而言的。如对于一个记录来说，它是所属文件的一个数据元素，而它相对于所含的数据项而言又可看做数据。因此，在本教材中，对数据和数据元素这两个术语的使用并不加以严格区别。

数据记录(Data Record)简称**记录**，它是数据处理领域组织数据的基本单位，数据中的每个数据元素在许多应用场合被组织成记录的结构。一个数据记录由一个或多个**数据项**(Item)所组成，每个数据项可以是简单数据项(即不可再分，如一个数值、一个字符等)，也可以是组合数据项(即数组或记录)。就拿对图书目录管理来说，每个记录表示一本图书的目录信息，如表 1-1 所示。

表 1-1 图书目录表

登录号	书 号	书 名	作 者	出版社	定 价
00001	ISBN 7-04-003907-9/TP·103	计算方法	唐 珍	高等教育	4.80
00002	ISBN 7-111-03247-0/TP·158	计算机辅助制造	李德庆	机械工业	3.30
00003	ISBN 7-04-005655-0/TP·312	C++ 程序设计基础	张基温	高等教育	17.00
00004	ISBN 7-302-00984-8/TP·363	数据结构	严蔚敏	清华大学	6.05
00005	ISBN 7-304-00543-2/TP·27	计算机应用基础	王 利	中央电大	8.05

（续表）

登录号	书　号	书　名	作　者	出版社	定　价
00006	ISBN 7-302-00860-4/TP·312	C 程序设计	谭浩强	清华大学	7.30
00007	ISBN 7-80046-822-4/O·026	应用概率统计	武继玉	航空航天	8.50
⋮	⋮	⋮	⋮	⋮	⋮

在表 1-1 中，第一行为表目行或目录行，它给出了该表中每条记录的结构。从表目行向下的每一行为一条记录，它给出了一本图书的有关信息；每一列为一个数据项，它描述了图书中的一种属性。每条记录由 6 个数据项组成，其名称分别为登录号、书号、书名、作者、出版社和定价，前五个数据项均为组合数据项——字符串，即字符数组，后一个数据项为简单数据项——浮点数。当记录不同时，所对应的同一数据项（属性）的值可能相同，也可能不同。例如对于第 4 条和第 6 条记录来说，出版社数据项的值相同，即为“清华大学”，而书名数据项的值不同，一个为“数据结构”，另一个为“C 程序设计”。当然，在一个表或文件中，不允许出现完全相同（即对应属性的值都相同）的两条记录。

在一个表或文件中，若所有记录的某个数据项的值都不同，也就是说，每个值能够唯一地标识一个记录时，则可把这个数据项作为记录的关键数据项，简称**关键项**（Key Item），关键项中的每一个值称做所在记录的**关键字**（Key Word 或 Key）。在表 1-1 中，登录号数据项的值都不同，所以可把登录号作为记录的关键项，其中的每一个值就是所在记录的关键字。如 00002 为第 2 条记录的关键字，00005 为第 5 条记录的关键字等。

在一个表或文件中，能作为关键项的数据项可能没有，可能只有一个，也可能多于一个。当没有时，可把多个有关的数据项联合起来，构成一个组合关键项，用组合关键项中的每一个组合值来唯一地标识一个记录，该组合值就是所在记录的关键字。

引入了记录的关键项和关键字后，为简便起见，在以后的讨论中，经常利用关键项来代替所有记录，利用关键字来代替所在的记录。

数据处理（Data Processing）是指对数据进行检索、插入、删除、合并、拆分、排序、统计、简单计算、转换、输入、输出等的操作过程。在早期，计算机主要用于科学和工程计算，进入 80 年代以后，计算机主要用于数据处理。据有关统计资料表明，现在计算机用于数据处理的时间比例平均高达 80% 以上，随着时间的推移和计算机应用的进一步普及，计算机用于数据处理的时间比例必将进一步增大。像计算机情报检索系统、经济信息管理系统、图书管理系统、物资调配系统、银行核算系统、财务管理系统等都是计算机在数据处理领域的具体应用。数据结构是进行数据处理的软件基础，因此，数据结构课程是计算机有关专业的主干课程之一。

数据结构（Data Structure），简单地说是指数据以及相互之间的联系。上面提到，数据的描述对象是现实世界的事物及其活动，而任何事物及其活动都不是孤立存在的，都是在一定意义上相互联系、相互影响的，所以数据之间必然存在着联系。数据之间的相互联系，被称为数据的**逻辑结构**。在计算机中存储数据时，不仅要存储数据本身，而且要存储它们之间的联系（即逻辑结构）。一种数据结构在存储器中的存储方式称为数据的**物理结构**或**存储结构**。由于存储方式有顺序、链接、索引、散列等多种形式，所以，一种数据结构可以根据应用

的需要表示成任一种或几种存储结构。数据的逻辑结构和存储结构都反映数据的结构，但通常所说的数据结构是指数据的逻辑结构，不包含存储结构的含义。

为了更确切地描述一种数据结构，通常采用二元组表示：

$$B = (K, R)$$

B 是一种数据结构，它由数据元素的集合 K 和 K 上二元关系的集合 R 所组成。其中

$$K = \{k_i \mid 1 \leqslant i \leqslant n,\ n \geqslant 0\}$$
$$R = \{r_j \mid 1 \leqslant j \leqslant m,\ m \geqslant 0\}$$

其中 k_i 表示集合 K 中的第 i 个数据元素，n 为 K 中数据元素的个数，特别地，若 $n = 0$，则 K 是一个空集，因而 B 也就无结构而言，有时也可以认为它具有任一结构；r_j 表示集合 R 中的第 j 个二元关系（以后均简称关系），m 为 R 中关系的个数，特别地，若 $m = 0$，则 R 是一个空集，表明集合 K 中的元素之间不存在任何关系，彼此是独立的，就象数学中集合里的元素一样。在本书所讨论的数据结构中，一般只讨论 $m = 1$ 的情况，即 R 中只包含一个关系（$R = \{r\}$）的情况。对于包含有多个关系的数据结构，可分别对每一个关系进行讨论。

K 上的一个关系 r 是序偶的集合。对于 r 中的任一序偶 $< x, y >$ $(x, y \in K)$，我们把 x 叫做序偶的第一元素，把 y 叫做序偶的第二元素，又称序偶的第一元素为第二元素的直接**前驱**（通常简称前驱），称第二元素为第一元素的直接**后继**（通常简称后继）。如在 $< x, y >$ 的序偶中，x 为 y 的前驱，而 y 为 x 的后继。

一种数据结构还能够利用图形形象地表示出来，图形中的每个结点（或叫顶点）对应着一个数据元素，两结点之间带箭头的连线（称作有向边或弧）对应着关系中的一个序偶，其中序偶的第一元素为有向边的起始结点，第二元素为有向边的终止结点，即箭头所指向的结点。

作为例子，下面根据表 1-2 构造出一些典型的数据结构。

表 1-2　教务处人事简表

职工号	姓　名	性　别	出生日期	职　务	部　门
01	万明华	男	1952.03.20	处长	
02	赵　宁	男	1958.06.14	科长	教材科
03	张　利	女	1954.12.07	科长	考务科
04	赵书芳	女	1962.08.05	主任	办公室
05	刘永年	男	1949.08.15	科员	教材科
06	王明理	女	1965.04.01	科员	教材科
07	王　敏	女	1962.06.28	科员	考务科
08	张　才	男	1957.03.17	科员	考务科
09	马立仁	男	1965.10.12	科员	考务科
10	邢怀常	男	1966.07.05	科员	办公室

表 1-2 中共有 10 条记录，每条记录都由 6 个数据项所组成，由于每条记录的职工号各不

相同，所以可把每条记录的职工号作为该记录的关键字，并在下面的例子中，我们将用记录的关键字来代表整个记录。

例 1 一种数据结构 set = (K,R)，其中

K = {01,02,03,04,05,06,07,08,09,10}

R = {}

在数据结构 set 中，只存在有元素的集合，不存在有关系的集合，表明我们只考虑表 1-2 中的每条记录，并不考虑它们之间的任何关系。具有此种特点的数据结构被称为**集合结构**。对于集合结构也可看做元素，按任一次序排列的线性结构，在存储器中可以根据需要按任一种存储方式进行存储。

例 2 一种数据结构 linearity = (K,R)，其中

K = {01,02,03,04,05,06,07,08,09,10}

R = {r}

r = { <05,01> , <01,03> , <03,08> , <08,02> , <02,07> , <07,04> ,
<04,06> , <06,09> , <09,10> }

对应的图形如图 1-1 所示。

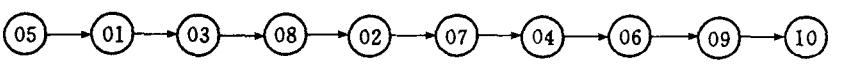

图 1-1 数据的线性结构示意图

结合表 1-2，细心的读者不难看出：r 是按职工年龄从大到小排列的关系。

在 linearity 中，每个数据元素有且仅有一个直接前驱元素(除结构中第一个元素 05 外)，有且仅有一个直接后继元素(除结构中最后一个元素 10 外)。这种数据结构的特点是数据元素之间的 1 对 1(1:1)联系，即**线性关系**，我们把具有这种特点的数据结构叫做**线性结构**。

例 3 一种数据结构 tree = (K,R)，其中

K = {01,02,03,04,05,06,07,08,09,10}

R = {r}

r = { <01,02> , <01,03> , <01,04> , <02,05> , <02,06> , <03,07> ,
<03,08> , <03,09> , <04,10> }

对应的图形如图 1-2 所示。

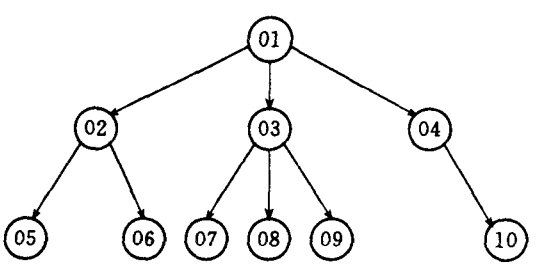

图 1-2 数据的树形结构示意图

结合表 1-2,细心的读者不难看出:r 是人员之间领导与被领导的关系。

图 1-2 像倒着画的一棵树,在这棵树中,最上面的一个没有前驱只有后继的结点叫做树根结点,最下面一层的只有前驱没有后继的结点叫做树叶结点,除树根和树叶之外的结点叫做树枝结点。在一棵树中,每个结点有且只有一个前驱结点(除树根结点外),但可以有任意多个后继结点(树叶结点可看做具有 0 个后继结点)。这种数据结构的特点是数据元素之间的 1 对 N(1:N)联系(N≥0),即**层次关系**,我们把具有这种特点的数据结构叫做**树形结构**,简称**树**。

例 4 一种数据结构 graph = (K,R),其中

$$K = \{01,02,03,04,05,06,07\}$$

$$R = \{r\}$$

$$r = \{<01,02>,<02,01>,<01,04>,<04,01>,<02,03>,<03,02>,<02,06>,<06,02>,<02,07>,<07,02>,<03,07>,<07,03>,<04,06>,<06,04>,<05,07>,<07,05>\}$$

对应的图形如图 1-3 所示。

从图 1-3 可以看出,r 是 K 上的对称关系,为了简化起见,我们把 <x,y> 和 <y,x> 这两个对称序偶用一个无序对(x,y)或(y,x)来代替;在图形表示中,我们把 x 结点和 y 结点之间两条相反的有向边用一条无向边来代替。这样 r 关系可改写为:

$$r = \{(01,02),(01,04),(02,03),(02,06),(02,07),(03,07),(04,06),(05,07)\}$$

对应的图形如图 1-4 所示。

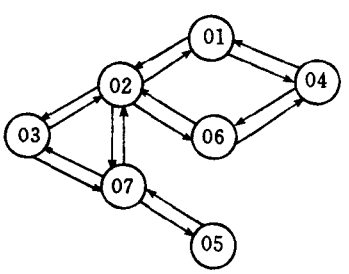

图 1-3 数据的图形结构示意图

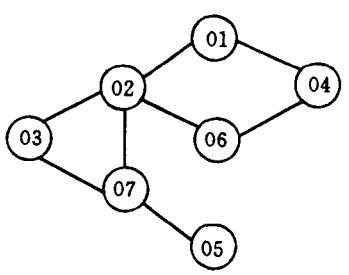

图 1-4 图 1-3 的等价表示

如果说 r 中每个序偶里的两个元素所代表的人员是好友的话,那么 r 关系就是人员之间的好友关系。

从图 1-3 或图 1-4 可以看出,结点之间的联系是 M 对 N(M:N)联系(M≥0,N≥0),即**网状关系**。也就是说,每个结点可以有任意多个前驱结点和任意多个后继结点。我们把具有这种特点的数据结构叫做**图形结构**,简称**图**。

从图形结构、树形结构和线性结构的定义可知,树形结构是图形结构的特殊情况(即 M=1的情况),线性结构是树形结构的特殊情况(即 N=1 的情况)。为了区别于线性结构,我们把树形结构和图形结构统称为**非线性结构**。

例 5 一种数据结构 B = (K, R), 其中

$K = \{k_1, k_2, k_3, k_4, k_5, k_6\}$

$R = \{r_1, r_2\}$

$r_1 = \{ < k_3, k_2 > , < k_3, k_5 > , < k_2, k_1 > , < k_5, k_4 > , < k_5, k_6 > \}$

$r_2 = \{ < k_1, k_2 > , < k_2, k_3 > , < k_3, k_4 > , < k_4, k_5 > , < k_5, k_6 > \}$

若用实线表示关系 r_1, 虚线表示关系 r_2, 则对应的图形如图 1-5 所示。

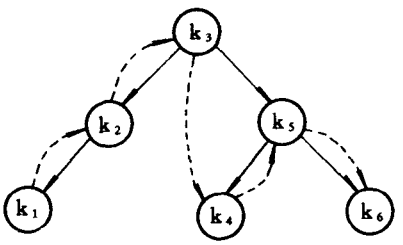

图 1-5 带有两个关系的一种数据结构示意图

从图 1-5 可以看出: 数据结构 B 是一种非线性的图形结构。但是, 若只考虑关系 r_1 则为树形结构, 若只考虑关系 r_2 则为线性结构。

数据类型(Data Type)是对数据的取值范围、每一数据的结构以及允许施加操作的一种描述。每一种计算机高级语言都定义有自己的数据类型。在通用的计算机高级语言中, 一般都具有整数、实数(浮点数)、枚举、字符、字符串、指针、数组、记录、文件等数据类型。如整数类型在计算机系统中通常用两个字节或四个字节表示, 若采用两个字节, 则整数表示范围在 $-2^{15} \sim 2^{15}-1$, 即 -32 768 ~ 32 767 之间; 若采用四个字节, 则整数表示范围在 $-2^{31} \sim 2^{31}-1$, 即 -2 147 483 648 ~ 2 147 483 647 之间。对整数类型的数据允许施加的操作通常有: 单目取正或取负运算, 双目加、减、乘、除、取模等运算, 双目等于、不等于、大于、大于等于、小于、小于等于等关系(比较)运算, 以及赋值运算。字符类型在机器中通常用一个字节表示, 无符号表示范围在 0 ~ 255 之间, 能够至多对 256 种字符进行编码, 对字符类型的数据允许进行的操作主要为赋值和各种关系运算。字符串类型为字符类型的顺序排列结构, 每一个字符的有限序列(其最大长度由具体语言规定)都是字符串类型中的一个值, 对字符串的操作主要有求串长度、串拷贝、两串连接、两串比较等。

数据类型可分为简单类型和结构类型两种。简单类型中的每个数据(即简单数据)都是无法再分割的整体, 如一个整数、实数、字符、指针、枚举量等都是无法再分割的整体, 所以它们所属的类型均为简单类型。结构类型由简单类型按照一定的规则构造而成, 并且结构类型仍可以包含结构类型, 所以一种结构类型中的数据(即结构数据)可以分解为若干个简单数据或结构数据, 每个结构数据仍可再分。如数组就是一种结构类型, 它由固定个数的同一类型顺序排列而成, 数组型中的每一个数组值包含有固定个数的同一类型数据, 每个数据(元素)都可以通过下标运算符直接访问。记录也是一种结构类型, 它由固定个数的不同(也可以相同)类型顺序排列而成, 记录型中的每一个记录值包含有固定个数的不同类型数据, 每个数据(域)都可以通过成员运算符直接访问。另外, 字符串和文件也都是结构类型。

无论是简单类型还是结构类型都有“型”和“值”的概念，一种数据类型中的任一数据称为该类型中的一个值（又称为实例），该值（实例）与所属数据类型具有完全相同的结构，数据类型所规定的操作就是在值上进行的，所以在一般的叙述中，并不明确指出是“型”还是“值”，读者应根据实际情况加以理解。如提到记录时，当讨论的是记录结构则认为是记录型，当讨论的是具体一条记录则认为是记录值。

数据类型也可以被定义为一种数据结构和对该数据结构允许进行的操作集。对于简单类型，其数据结构就是相应取值范围内的所有数据，每一个数据值是不可分的独立整体，因而数据值内部就无结构而言。对于结构类型，其数据结构就是相应元素的集合（它实际上是该结构类型中的一个值）和元素之间所含关系的集合。如对于常用的数组、记录、字符串和文件等结构类型来说，其元素都是按位置有序排列的，就此可把它们均看做线性数据结构。数组的数据结构可描述如下：

$$array = (A, R)$$

其中

$$A = \{a_i \mid 0 \leqslant i \leqslant n-1, n \geqslant 1\}$$

$$R = \{r\}$$

$$r = \{ < a_i, a_{i+1} > \mid 0 \leqslant i \leqslant n-2\}$$

从以上描述可知：a_i 为数组中的第 i 个元素，n 为大于等于 1 的整数，用来表明数组中元素的个数，数组元素的下标从 0 ~ n - 1，数组中前后相邻位置上的两个元素为一个序偶，其前一元素 a_i 是后一元素 a_{i+1} 的前驱，而 a_{i+1} 是 a_i 的后继，第一个元素 a_0 无前驱元素，最后一个元素 a_{n-1} 无后继元素。

当数组是一个二维数组时，它是一个嵌套的一维数组，可把它看做是按行位置有序的线性结构，该结构中的元素为一维数组，每一行又是按列位置有序的线性结构，该结构中的元素为具有数组基本类型的数据，该数据通过行和列下标直接访问。当数组具有三维或更多维时，也可进行类似的分析。

抽象数据类型（Abstract Data Type，ADT）由一组数据结构和在该组数据结构上的一组操作所组成。由于数据结构隐含在数据之中（如数组隐含元素之间的线性关系），所以抽象数据类型也可以被定义为由一组数据和在该组数据上的一组操作所组成。抽象数据类型包含一般数据类型的概念，但含义比一般数据类型更广、更抽象。一般数据类型由具体语言系统内部定义，直接提供给编程者定义用户数据（但对于结构类型，编程者要给出结构名和成员定义），因此称它们为预定义数据类型。抽象数据类型通常由编程者定义，包括定义它所使用的数据（数据结构）和在这些数据上所进行的操作。在定义抽象数据类型中的数据部分和操作部分时，要求只定义到数据的逻辑结构和操作说明，不考虑数据的存储结构和操作的具体实现（即具体操作代码），这样抽象层次更高，更能为其他用户提供良好的使用接口。

抽象数据类型在 C ++ 语言中是通过类类型来描述的，其数据部分通常定义为类的私有（private）或保护（protected）的数据成员，它只允许给该类或派生类直接使用，操作部分通常定义为类的公共（public）的成员函数，它既可以提供给该类或派生类使用也可以提供给外部操作使用，操作部分只给出操作说明（即函数声明），操作的具体实现通常在一个单独文件中给出，使它与类的定义（即声明）相分离，类的声明通常被存放在一个专门的头文件（其扩展名为 .h）中，这样能够实现信息的隐藏、封装、重用和继承，符合面向对象程序设计（Object-Ori-

ented Programming, OOP)的思想。

在本书中,考虑到读者对 C++ 中的类类型可能不太熟悉,所以在定义抽象数据类型时,不是采用类类型,而是采用记录类型(即 C/C++ 中的 struct 类型)定义数据部分;不是采用类类型中的成员函数,而是采用外部函数(即普通函数)定义(即实现)操作部分中的每一个操作。有时在抽象数据类型的数据部分,也不是给出抽象描述,即数据的逻辑结构描述,而是直接给出数据的类型定义,此时其数据不仅含有逻辑结构,也含有存储结构,这样,数据的类型确定后就能够编写出每一个操作的具体实现。虽然本书没有采用类类型来描述抽象数据类型,但读者只要能够掌握本书所介绍的每一种抽象数据类型及其实现的方法,在学会了类类型的使用方法后,不难把每一种抽象数据类型用类类型来描述和实现。

在本书中,描述一种抽象数据类型将采用如下书写格式:

```
ADT <抽象数据类型名> is
    Data:
        <数据描述>
    Operations:
        <操作声明>
end <抽象数据类型名>
```

例如设计矩形的一种抽象数据类型,其数据部分包括矩形的长度和宽度,操作部分包括初始化矩形的尺寸、求矩形的周长和求矩形的面积。

假定该抽象数据类型名用 RECtangle(矩形)表示,定义矩形长度和宽度的数据用 length 和 width 表示,并假定其类型为浮点(float)型,初始化矩形数据的函数名用 InitRectangle 表示,求矩形周长的函数名用 Circumference(周长)表示,求矩形面积的函数名用 Area(面积)表示,则矩形的 ADT(抽象数据类型)描述如下:

```
ADT RECtangle is
    Data:
        float length, width;
    Operations:
        Rectangle InitRectangle(float len, float wid);
        float Circumference(Rectangle r);
        float Area(Rectangle r);
end RECtangle
```

下面讨论该抽象数据类型的具体实现,这里矩形的记录(在 C/C++ 中称为结构)类型名用 Rectangle 表示,该类型的定义如下:

```
struct Rectangle{
    float length, width;
};
```

初始化矩形数据的函数定义如下:

```
Rectangle InitRectangle(float len, float wid){
    Rectangle r;  //定义局部变量 r
```

```
    r.length = len;  //把 len 值赋给 r 的 length 域
    r.width = wid;  //把 wid 值赋给 r 的 width 域
    return r;  //返回 r
}
```

该函数带有两个值参数 len 和 wid,分别接受调用函数传送来的矩形的长度和宽度,该函数返回一个 Rectangle 类型的数据给调用函数,在调用函数中使之赋给一个该类型的变量,此变量就被初始化了。求矩形周长和求矩形面积的函数分别定义如下:

```
float Circumference(Rectangle r){
    return 2 * (r.length + r.width);
}

float Area(Rectangle r){
    return r.length * r.width;
}
```

这两个函数分别具有一个矩形值参数,调用执行后分别计算并返回被传送矩形的周长和面积。

用 C/C++ 语言编写出完整的程序如下:

```
//程序 1-1.cpp
#include<iostream.h>
struct Rectangle{
    float length, width;
};

Rectangle InitRectangle(float len, float wid); //超前声明
float Circumference(Rectangle r); //超前声明
float Area(Rectangle r); //超前声明

void main(void){
    float x, y;  //用于从键盘上输入一个矩形的长和宽
    float p, s;  //用于保存矩形的周长和面积
    Rectangle r;  //定义一个矩形结构(记录)变量
    cout<<"请输入一个矩形的长和宽!"<<endl;
    cin>>x>>y;
    r = InitRectangle(x, y);
    p = Circumference(r);
    s = Area(r);
    cout<<endl;
    cout<<"矩形的周长为:"<<p<<endl;
    cout<<"矩形的面积为:"<<s<<endl;
}
```

```
Rectangle InitRectangle(float len, float wid){
    Rectangle r;
    r.length = len;
    r.width = wid;
    return r;
}

float Circumference(Rectangle r){
    return 2 * (r.length + r.width);
}

float Area(Rectangle r){
    return r.length * r.width;

}
```

假定程序运行后从键盘上输入的一个矩形的长和宽的尺寸分别为 4 和 5,则得到的运行结果如下:

```
请输入一个矩形的长和宽!
4 5

矩形的周长为:18
矩形的面积为:20
```

数据对象(Data Object)简称**对象**,它属于一种数据类型(包含一般数据类型和抽象数据类型)中的特定量(又称实例),包括常量和变量。如 25 为一个整型数据对象,'A'为一个字符数据对象,语句 char * p 定义 p 为一个字符指针(即字符串)对象,int a[10]定义 a 为一个含有 10 个整型数的数组对象,Rectangle r1 定义 r1 为一个 Rectangle 结构(记录)类型的对象,RECtangle rec 定义 rec 为一个具有 RECtangle 抽象数据类型的对象。

算法(Algorithm)就是解决特定问题的方法。描述一个算法可以采用文字叙述,也可以采用传统流程图、N - S 图或 PAD 图,但要在计算机上实现,则最终必须采用一种程序设计语言编写为程序。作为一个算法应具备以下 5 个特性:

(1) 有穷性　一个算法必须在执行有穷步之后结束。

(2) 确定性　算法中的每一步都必须具有确切的含义,无二义性。

(3) 可行性　算法中的每一步都必须是可行的,也就是说,每一步都能够通过手工或机器可以接受的有限次操作在有限时间内实现。

(4) 输入　一个算法可以有 0 个、1 个或多个输入量,在算法被执行之前提供给算法。

(5) 输出　一个算法执行结束后至少要有一个输出量,它是利用算法对输入量进行运算和处理的结果。

需要人们解决的特定问题可分为数值的和非数值的两类。解决数值问题的算法叫做数值算法,科学和工程计算方面的算法都属于数值算法,如求解数值积分,求解线性方程组,求解代数方程,求解微分方程等。解决非数值问题的算法叫做非数值算法,数据处理方面的算

法都属于非数值算法，如对各种数据结构的排序算法、查找算法、插入算法、删除算法、遍历算法等。数值算法和非数值算法并没有严格的区别，一般说来，在数值算法中主要进行算术运算，而在非数值算法中，则主要进行比较和逻辑运算。另一方面，特定的问题可能是递归的，也可能是非递归的，因而解决它们的算法就有递归算法和非递归算法之分。当然，从理论上讲，任何递归算法都可以通过循环、堆栈等技术转化为非递归算法。

在计算机领域，一个算法实质上是针对所处理问题的需要，在数据的逻辑结构和存储结构的基础上施加的一种运算。由于数据的逻辑结构和存储结构不是唯一的，在很大程度上可以由用户自行选择和设计，所以处理同一个问题的算法也不是唯一的。另外，即使对于具有相同的逻辑结构和存储结构而言，其算法的设计思想和技巧不同，编写出的算法也大不相同。我们学习数据结构这门课程的目的，就是要学会根据数据处理问题的需要，为待处理的数据选择合适的逻辑结构和存储结构，进而按照结构化、模块化以及面向对象的程序设计方法设计出比较满意的算法(程序)。

1.2 算法描述

算法就是解决特定问题的方法，该方法可以借助各种工具描述出来。如从 n 个整数元素中查找出最大值，若用流程图描述则如图 1-6 所示。

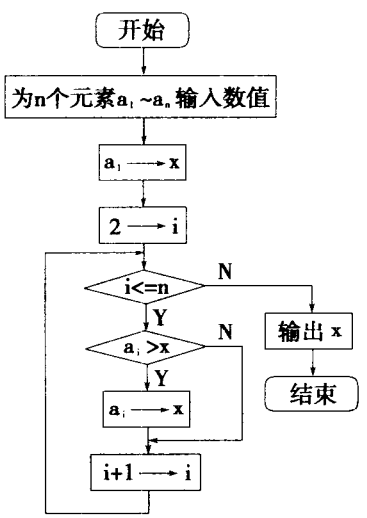

图 1-6 求 n 个元素中的最大值

若采用文字描述，则如下列步骤所示：

(1) 给 n 个元素 $a_1 \sim a_n$ 输入数值；

(2) 把第一个元素 a_1 赋给用于保存最大值元素的变量 x；

(3) 把表示下标的变量 i 赋初值 2；

(4) 如果 i <= n 则向下执行，否则输出最大值 x 后结束算法；

(5) 如果 $a_i > x$ 则将 a_i 赋给 x，否则不改变 x 的值，这使得 x 始终保存着当前比较过的所有元素的最大值；

(6) 使下标 i 增 1,以指示下一个元素;

(7) 转向第(4)步继续执行。

若要使一个算法在计算机上实现,则最终必须采用一种程序设计语言进行描述。如对于上述算法,采用 C++ 语言描述如下:

```
//程序 1-2.cpp
#include<iostream.h>
const n=10;  //假定 n 等于 10
void main(void)
{
    int i, x, a[n];  //用 a[0]~a[n-1]保存 a1~an 元素
    cout<<"请输入 10 个整数:";
    for(i=0; i<n; i++)
        cin>>a[i];
    x=a[0]; i=1;
    while(i<n){
        if(a[i]>x) x=a[i];
        i++;
    }
    cout<<"10 个整数中的最大值为:"<<x<<endl;
}
```

本书对所有算法一般采用文字和 C++ 语言两种描述,文字描述给出算法的思路和执行步骤,C++ 语言描述给出在机器上实现的代码。下面对 C++ 语言的有关内容做简要说明,为以后分析和编写算法做准备。

1.2.1 包含文件语句

包含文件语句是以关键字 #include 开头,后跟用尖括号或双引号括起来的头文件名,行后不需要使用分号。下面介绍几个常用的系统头文件的作用。

1. #include<iostream.h>

在程序的开始使用该语句后,在其后的每一个函数中,都可以使用标准输入设备(键盘)流对象 cin,标准输出设备(屏幕)流对象 cout 和标准错误输出设备(屏幕)流对象 cerr ,以及使用用于输入的提取操作符 >> 和用于输出的插入操作符 << 进行数据输入输出操作。对于基本类型为 char、short、int、long、char *(字符串型)、float、double、long double 的数据能够直接进行输入和输出;对于非字符指针类型的指针型数据能够直接输出指针(即操作数地址);对于其他类型的数据,只有通过对 >> 和 << 操作符重载后才能直接输入和输出,当然若数据中的元素为基本数据类型,则可对其元素直接输入和输出。例如,一种记录结构类型如下:

```
struct worker {
    int id;
    char name[20];
    float wage;
}
```