数据结构简明教程

（第2版）

徐孝凯 王凤禄 编著

清华大学出版社

数据结构简明教程

（第 2 版）

徐孝凯　王凤禄　编著

清华大学出版社
北　京

内 容 简 介

本书是为数据结构的初学者编写的。书中详细介绍了集合、线性表、栈、队列、二叉树、二叉搜索树、堆、图等具体而常用的数据结构，介绍了对这些数据结构建立顺序、链接、索引、散列等相应存储结构的方法和算法，介绍了在数据存储结构的基础上对数据进行查找、插入、删除、修改、排序、遍历等运算的方法和算法，以及相应的时间和空间复杂度。本书的每章均给出了丰富的练习题，书后附有部分习题的参考解答。

本书的主要读者对象为在校计算机专业专科（含高职）、非计算机专业本科和专科学习数据结构课程的学生。

图书在版编目(CIP)数据

数据结构简明教程/徐孝凯，王凤禄编著. —2 版. —北京：清华大学出版社，2005.5
ISBN 7-302-10610-X

I. 数… II. ①徐… ②王… III. ①数据结构-高等学校-教材 ②C 语言-程序设计-高等学校-教材 IV. TP311.12

中国版本图书馆 CIP 数据核字（2005）第 017658 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
责任编辑：郑寅堃
印 刷 者：北京市世界知识印刷厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：14.75　**字数**：362 千字
版　　次：2005 年 5 月第 2 版　2005 年 5 月第 1 次印刷
书　　号：ISBN 7-302-10610-X/TP・7196
印　　数：1～4000
定　　价：19.00 元

导　读

清华大学出版社出版的《数据结构简明教程》一书，自 1995 年出版以来，已重印过十几次，被许多院校计算机及相关专业用为教材。由于原教材采用 PASCAL 语言描述，而现在使用较多的是 C 和 C++语言，所以这次用 C++语言重新编写了此书，并且按照目前对数据结构课程的教学要求进行了全面的内容更新和调整，更加注重其简明性、实用性和科学性。希望再版后能够继续受到读者的关注和好评。

写作特点

- 本书是以初次学习数据结构知识的读者为对象而编写的，处处注意体现教材的编写特点和风格。
- 本书内容丰富实用，概念定义准确，叙述深入浅出，结构层次分明，算法分析透彻，章节安排连贯有序，便于阅读和自学。全书中的每种数据结构和相应的算法举例都经过精心设计，具有典型性和代表性，为读者进行更复杂的数据结构设计和算法设计奠定了坚实的基础。
- 在每章开始有学习目标，给出读者学习本章后应掌握的知识点；每章正文之后有小结，对本章内容进行了全面和系统的总结和归纳，帮助读者加深对所学内容的理解和认识；每章最后给出了丰富的练习题，读者通过做练习和上机操作实践，能够大大提高程序设计和软件开发能力；书后附有部分习题的参考解答。这些措施有助于学好数据结构这门课程。
- 本书中所有算法和程序都在 Microsoft Visual C++ 6.0 集成开发环境下运行通过。

读者对象

- 在校计算机专业专科（含高职）、非计算机专业本科和专科学习数据结构课程的学生。
- 需要自学数据结构知识或需要加深学习 C++ 编程知识的有关人员。
- 参加各种计算机知识培训、特别是软件知识培训的学员。

本书也可作为学习参考书使用。

学习基础

具有 C、C++、Java、VB 等任一种程序设计语言基础，最好具有 C++语言基础。

学习目标

学习本书后能够为学习后续所有计算机课程打下程序设计和软件开发基础。

作者联系方式

若在学习中碰到问题需要同作者联系，或者有什么好的建议需要同作者交流，可拨打电话 010-64910302 或发电子邮件到 xuxk@crtvu.edu.cn。作者真诚欢迎与广大读者交流，并恳切希望得到各种宝贵意见。

徐孝凯　王凤禄

2004 年 8 月

目　录

第1章 绪 论

学习目标

本章主要介绍数据结构技术中一些常用的概念、算法描述方法以及算法评价标准等内容。通过本章学习，要求：

- 掌握集合、线性、树、图等数据结构的特点；
- 掌握抽象数据类型的定义以及实现方法；
- 了解算法的一般特性和算法描述的方法；
- 掌握评价算法的一般标准，算法的时间复杂度的数量级表示，算法的空间复杂度的数量级表示；
- 掌握各种不同数量级随着解决问题规模的增大而变化的趋势，以及相互比较结果。

数据结构是从事计算机教学、研究、开发、应用等工作所必须学习和掌握的一门基础知识。它专门研究由现实世界抽象出来的数据在计算机系统中的表示、组织、存储和处理的方法。

用计算机存储数据不仅要存储数据本身，而且要存储数据之间的各种联系（即数据结构）。存储具有四种基本结构，即顺序、链接、散列和索引，由它们可以根据需要组合成其他任何复杂的存储结构。

对数据进行处理的方法（算法）是根据人们解决实际问题的需要而逐渐产生、发展和丰富起来的。现在人们已经总结出各种通用而且有效的算法，掌握这些算法是进行各种软件开发和设计的基础。要让计算机执行一个算法，必须采用一种计算机语言（如 C 或 C++ 语言）加以描述，所以学习数据结构知识之前必须学会一种计算机语言，这样才能够上机调试和运行算法，检测算法的各种性能。

1.1 常用术语

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。例如，一个人的名字可以用一个字符串来描述，一条曲线可以用一个数组来描述，数组中的每一个元素用来存储曲线中对应点的坐标值和颜色编号。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。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。

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，每个下标位置上的值就是它的数据元素。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是数据，数据项是它的一个成分（元素）。因此，在本书中，对数据和数据元素这两个术语的使用不加以严格区别。

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、字符串或记录）。就拿对图书目录的管理来说，每个记录表示一本图书的有关信息，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 7-302-04632-8/TP・2745	C++语言基础教程	徐孝凯	清华大学	26.00
00002	ISBN 7-03-008966-9/O・1313	离散数学	蔡经球	科学	22.00
00003	ISBN 7-5053-8168-7/TP・4757	计算机系统原理	张基温	电子工业	25.00
00004	ISBN 7-304-01979-X/TP・147	数据结构	许卓群	中央电大	28.00
00005	ISBN 7-304-02494-1/TP・192	数据库基础与应用	刘世峰	中央电大	28.00
00006	ISBN 7-5053-5607-0/TP・2864	现代网络技术教程	张公忠	清华大学	34.00
00007	ISBN 7-04-006948-2/TP・52	软件开发基础	庞丽萍	高等教育	22.60
.....	.....	.....	.....	.....	.....

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。表1-1的每条记录由六个数据项组成，其名称分别为登录号、书号、书名、作者、出版社和定价，前五个数据项均为字符串，后一个数据项为实数。

在一个表中，若所有记录的某个数据项的值均不同，也就是说，每个值能够惟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），关键项中的每个值称为所在记录的**关键字**（key word 或 key）。在表1-1中，登录号数据项的值均不同，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为第二条记录的关键字，00005为第五条记录的关键字等。

在一个表中，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惟一地标识一个记录，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，以后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，而把记录中的其他项忽略掉。

数据结构（data structure）是指数据及其相互之间的联系。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处理的需要表示成任一种存储结构或它们的组合结构。数据的逻辑结构和存储结构是反映在不

同层面（即现实世界层面和计算机世界层面）上数据的结构，通常都把它称为数据结构，具体含义应结合上下文理解。

为了更确切地描述数据的逻辑结构，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，此时 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，表明不考虑集合 K 中元素之间存在的任何关系，彼此是独立的，就像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（$R=\{R_1\}$，通常直接把这个关系用 R 表示）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**前驱**，称第二元素为第一元素的**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示意地表示出来，示意图中的每个结点（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	1962.03.20	处长	
02	赵　宁	男	1968.06.14	科长	教材科
03	张　利	女	1964.12.07	科长	考务科
04	赵书芳	女	1972.08.05	主任	办公室
05	刘永年	男	1959.08.15	科员	教材科
06	王明理	女	1975.04.01	科员	教材科
07	王　敏	女	1972.06.28	科员	考务科
08	张　才	男	1967.03.17	科员	考务科
09	马立仁	男	1975.10.12	科员	考务科
10	邢怀常	男	1976.07.05	科员	办公室

表 1-2 中共有 10 条记录，每条记录都由六个数据项所组成，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，并且在下面的例子中，我们将用记录的关键字来代表整个记录。

例 1-1　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，只存在着元素的集合，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。我们称具有此种特点的数据结构为**集合结构**。对于集合结构，元素之间按任何次序排列都是允许的，如可以按关键字的升序排列，也可以按关键字的降序排列，等等，这可根据处理问题的需要任意决定。

例 1-2 一种数据结构 linearity=(K, R)，其中

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

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。

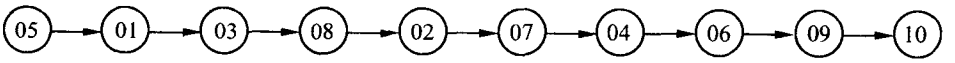

图 1-1 数据的线性结构示意图

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线性关系。

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，被称为**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 1-3 一种数据结构 tree= (K, R)，其中

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

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

对应的示意图如图 1-2 所示。

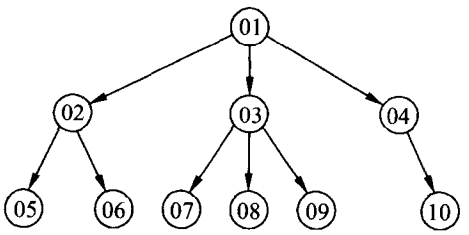

图 1-2 数据的树形结构示意图

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，最上面一层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（$N \geqslant 0$），被称为**层次关系**，我们把具有这种特点的数据结构叫做**树结构**，简称**树**。

例 1-4　一种数据结构 graph=(K, R)，其中

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

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；在示意图中，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。

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），被称为**网状关系**，也就是说，每个结点可以有任意个前驱结点和任意个后继结点。我们把具有这种特点的数据结构叫做**图结构**，简称**图**。

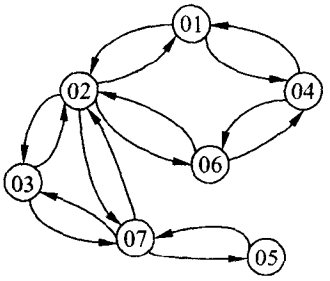

图 1-3　数据的图形结构示意图

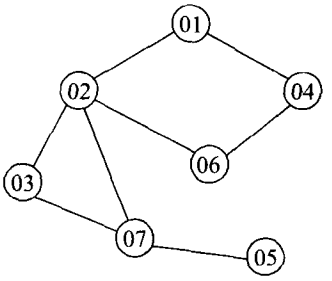

图 1-4　图 1-3 的等价表示

由图结构、树结构和线性结构的定义可知，树结构是图结构的特殊情况，线性结构又是树结构的特殊情况，当然更是图结构的特殊情况。为了区别于线性结构，我们把树结构和图结构统称为**非线性结构**。线性结构和非线性结构相对于集合来说都是有结构的，而集合结构为无结构的，即元素之间无任何联系。

例 1-5　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。

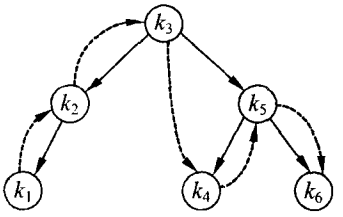

图 1-5　带有两个关系的数据结构示意图

从图 1-5 可以看出：数据结构 B 是一种非线性的图结构。但是，若只考虑关系 R_1 则为树结构，若只考虑关系 R_2 则为线性结构。

当一组数据中包含有多个关系时，通常把它们看做多个数据结构来分别分析和处理。

如对于表 1-2，根据处理不同问题的需要，就可能存在着按年龄排列建立的线性关系、按领导和被领导建立的层次关系，按好友联系建立的网状关系等，它们同数据集分别构成相应的数据结构。

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或四个字节表示。若采用两字节，则整数表示范围在-2^{15}~$2^{15}-1$，即-32768~32767之间；若采用四字节，则整数表示范围在-2^{31}~$2^{31}-1$，即-2147483648~2147483647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、串复制、串连接、串比较等运算。

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）通常只作为整体使用，如一个整数、实数、字符、指针、枚举量等。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数据顺序排列而成，数组类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）类型数据顺序排列而成，记录类型中的每一个记录值包含有固定个数的不同类型数据，每个数据都可以通过成员运算符直接访问。另外，像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”的区别，“型”是“值”的抽象定义，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”，读者应根据上下文加以理解。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”。

对于在计算机语言中使用的数组、记录、字符串和文件等结构类型，每个值中的元素可看做是按位置有序排列的，据此均认为它们具有线性结构。数组的数据结构可描述如下：

array=(A, R)，其中

$A=\{a[i] \mid 0\leqslant i\leqslant n-1, n\geqslant 1\}$

$R=\{<a[i], a[i+1]> \mid 0\leqslant i\leqslant n-2\}$

$a[i]$为数组中的下标为 i 的元素，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到 $n-1$，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。

按数组下标的个数，可把数组分为一维、二维、三维等。一维数组中的每个元素只包

含有一个下标；二维数组中的每个元素包含有两个下标，第一个称为行下标，第二个称为列下标。

二维数组可看做是一维数组的推广或嵌套，即首先把它看做是按行下标顺序排列的一维数组，该数组中的每个元素又都是按列下标顺序排列的一维数组。如对于一个二维数组 $b[m][n]$，可看做为一维数组 $b[m]$，所含元素依次为 $b[0], b[1], \cdots, b[m-1]$，其中每一个元素 $b[i]$（$0 \leqslant i \leqslant m-1$）又都是一个含有 n 个元素的一维数组，所含元素依次为 $b[i][0], b[i][1], \cdots, b[i][n-1]$。

同样，三维数组包含有三个下标，每个元素的位置由一组三个下标值惟一确定。三维数组是一维数组的三层嵌套结构。如对于一个三维数组 $c[p][m][n]$，首先可看做是一维数组 $c[p]$，所含元素依次为 $c[0], c[1], \cdots, c[p-1]$，其中每一个元素 $c[k]$（$0 \leqslant k \leqslant p-1$）又都是一个含有 m 个元素的一维数组，所含元素依次为 $c[k][0], c[k][1], \cdots, c[k][m-1]$，这里的每一个元素 $c[k][i]$（$0 \leqslant i \leqslant m-1$）也都是一个含有 n 个元素的一维数组，所含元素依次为 $c[k][i][0], c[k][i][1], \cdots, c[k][i][n-1]$。

数组的存储结构是顺序结构，即数组中第 $i+1$ 个元素紧接着存储在第 i 个元素的存储位置的后面。如对于一维数组 $a[n]$，则每个元素 $a[i]$的存储位置的首字节地址为：

$$\text{Address}(a[i])=\text{Loc}(a)+i \cdot L \qquad (0 \leqslant i \leqslant n-1)$$

其中 Loc(a)表示数组 a 的存储空间的首地址，L 表示数组 a 中元素类型的大小，即每个元素所占用的字节数，可用 sizeof($a[i]$)计算出来。由上述公式可知：元素 $a[0]$的存储地址为 Loc(a)，它就是整个数组的开始地址，$a[1]$的存储地址为 Loc(a)+1 · L，$a[2]$ 的存储地址为 Loc(a)+2 · L, $\cdots$, $a[n-1]$的存储地址为 Loc(a)+(n–1) · L。

对于一个二维数组 $b[m][n]$，每一行元素 $b[i]$的存储位置（即存储该行 n 个元素的首字节地址）为：

$$\text{Address}(b[i])=\text{Loc}(b)+i \cdot RS \qquad (0 \leqslant i \leqslant m-1)$$

其中 Loc(b)表示二维数组 b 的存储空间的首地址，RS 表示顺序存储一行 n 个元素所占用存储空间的大小，它等于每个元素所占用的字节数 L 与一行上元素的个数 n 的乘积。因此上述计算公式可改写为：

$$\text{Address}(b[i])=\text{Loc}(b)+i \cdot n \cdot L \qquad (0 \leqslant i \leqslant m-1)$$

对于二维数组 b 中的第 i 行（即下标为 i 的行），其中下标为 j 的元素 $b[i][j]$的存储位置为：

$$\text{Address}(b[i][j])=\text{Loc}(b)+i \cdot n \cdot L+j \cdot L \qquad (0 \leqslant i \leqslant m-1, 0 \leqslant j \leqslant n-1)$$

对于三维或更高维数组，读者可进行类似的分析。

抽象数据类型（abstract data type，ADT）由一组数据和在该组数据上的操作集所组成。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。抽象数据类型包括数据定义和操作定义两个部分，具体可以采用如下格式进行描述。

```
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>
```

在面向过程的程序设计中，对抽象数据类型的数据部分通常采用计算机语言中提供的简单类型、数组类型或记录（结构）类型来描述，对其操作部分通常采用计算机语言中提供的函数（过程）定义来描述；在面向对象的程序设计中，抽象数据类型与计算机语言中提供的类定义相对应。

下面看一个简单的例子。

我们可以把矩形定义为一种抽象数据类型，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:
        void InitRectangle(Rectangle& r, float len, float wid);
        float Circumference(Rectangle& r);
        float Area((Rectangle& r);
end RECtangle
```

其中参数 *r* 的类型名 Rectangle 表示一个用户定义的记录（结构）类型，它包括矩形的长度和宽度两个域，该记录类型用来统一描述此抽象数据类型所含的数据部分，用 C/C++ 语言定义如下：

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

初始化矩形数据的函数定义如下：

```
void InitRectangle(Rectangle& r, float len, float wid) {
    r.length=len;              //把 len 值赋给 r 的 length 域
    r.width=wid;               //把 wid 值赋给 r 的 width 域
}
```

该函数把两个值参数 len 和 wid 的值分别赋给引用参数 *r* 的 length 域和 width 域，实现对一个矩形 *r* 的初始化。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引用参数(也可采用值参)，调用执行后分别计算并返回被引用矩形的周长和面积。

在函数参数中，有引用参数和值参数之分，若在参数类型和参数名之间使用&符号，则就定义该参数为引用参数，否则为值参数。对于引用参数，函数被调用时，它被看做是对应的调用参数（即实参）的别名，函数中访问它就是访问对应的实参；对于值参数，当函数被调用时，将为它分配存储空间，并用对应实参的值初始化，在函数体中对值参的访问与对应的实参无关，当函数调用结束后将自动释放为值参所分配的存储空间。

若采用C++类来描述抽象数据类型RECtangle，则如下所示。

```
class RECtangle {
    private:
        float length, width;
    public:
        RECtangle(float len, float wid) {
            length=len; width=wid;
        }
        float Circumference(void) {
            return 2*(length+width);
        }
        float Area(void) {
            return length*width;
        }
};
```

这里，为了更符合类的定义，把原来的初始化函数改用构造函数实现。

下面给出使用此类的完整程序。

```
#include<iostream.h>
class RECtangle {
    //类的定义体同上
};
void main(void) {
    float x, y;                 //用于保存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
    cout<<"请输入一个矩形的长和宽!"<<endl;
    cin>>x>>y;
    RECtangle a(x,y);           //定义一个矩形类对象a
    p=a.Circumference();        //计算矩形a的周长
    s=a.Area();                 //计算矩形a的面积
    cout<<"矩形的周长为:"<<p<<endl;
```

```
    cout<<"矩形的面积为:"<<s<<endl;
}
```

数据对象（data object）简称**对象**，它属于一种数据类型中的特定实例。如25为一个整型数据对象，'A'为一个字符数据对象，char * p定义p为一个字符指针对象，它可以用来指向一个字符串，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r_1为一个类对象。

算法（algorithm）就是解决特定问题的思路和方法。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图等描述，但要在计算机上实现，则最终必须采用一种程序设计语言来描述。作为一个算法应具备以下五个特性：

（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和非数值两类。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。在各种数据结构上进行的排序算法、查找算法、插入算法、删除算法、遍历算法等均为非数值算法。数值算法和非数值算法并没有严格的区别，一般来说，在非数值算法中，主要进行比较和逻辑运算，而在数值算法中则主要进行各种算术运算。

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。

1.2 算法描述

一个算法可以借助各种表示工具描述出来。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。

若采用文字描述，则如下列步骤所示：

（1）给n个元素a_1~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；

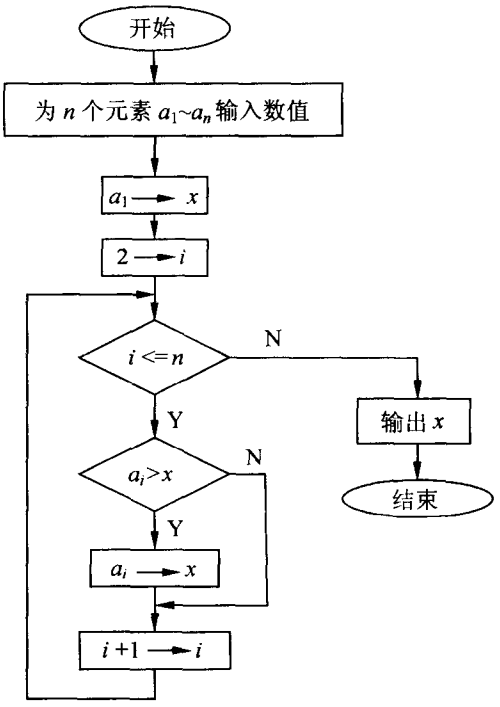

图 1-6　求 n 个元素中的最大值

（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++语言描述则可以编写成如下程序。

```
#include<iostream.h>
const int nn=10;                    //定义符号常量 nn 等于 10
int FindMax(int a[], int n)         //求数组 a 中几个元素的最大值
{
    int i, x;
    x=a[0];                         //把第一个元素 a[0]的值赋给 x
    i=1;                            //把第二个元素 a[1]的下标 1 赋给 i
    while(i<n){
        if(a[i]>x) x=a[i];
        i++;
    }
    return x;                       //返回数组 a 中 n 个元素的最大值
}

void main(void)
{
    int i, x, a[nn];                //用 a[0]~a[nn-1]保存 a1~ann 元素
    cout<<"请输入"<<nn<<"个整数: ";
```