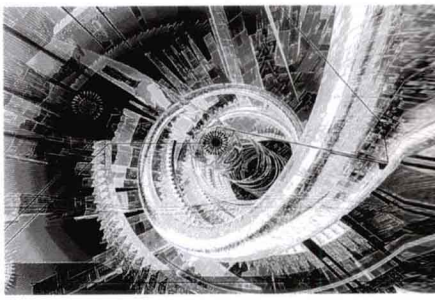
工业和信息化人才培养规划教材
高职高专计算机系列

数据结构

（第2版）



Data Structures

宗大华 陈吉人 ◎ 编

精心选取内容，难易适中，概念和算法都配以例题作解释、说明，帮助读者正确理解。从算法描述、算法分析和算法讨论三方面详解每一个算法，加快理解和掌握，并使读者从中感悟到程序编写的技巧和方法。

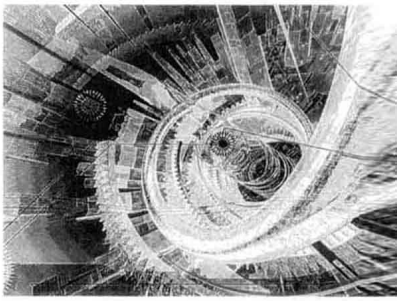
人民邮电出版社
POSTS & TELECOM PRESS

精品系列

工业和信息化人才培养规划教材 高职高专计算机系列

数据结构

（第2版）



Data Structures

宗大华 陈吉人 ◎ 编

人民邮电出版社
北京

图书在版编目（CIP）数据

数据结构 / 宗大华，陈吉人编. -- 2版. -- 北京 : 人民邮电出版社，2013.5
工业和信息化人才培养规划教材. 高职高专计算机系列
ISBN 978-7-115-30807-8

Ⅰ. ①数… Ⅱ. ①宗… ②陈… Ⅲ. ①数据结构－高等职业教育－教材 Ⅳ. ①TP311.12

中国版本图书馆CIP数据核字(2013)第019468号

内 容 提 要

本书是专门为高职高专计算机专业学生编写的数据结构教材。全书共9章，分为3大部分：第一部分（第1章）是对数据结构的概述，是学习本书的基础；第二部分（第2章到第7章）逐一介绍各种数据结构、存储实现及其常见算法；第三部分（第8章和第9章）介绍查找技术和排序技术。

“数据结构”是一门重要的专业基础课程。基于数据结构课程本身理论性、抽象性较强的特点，以及当前高职高专学生的认知能力和水平，本书在编写过程中尽力做到精心选取内容，并配以大量例题和习题（共有例题95个、习题278个），对给出的大多数算法都从“算法描述”、“算法分析”和“算法讨论”3个方面进行讲述，使学生能更好地理解算法，更快地掌握算法，希望学生能够从中感悟到程序编写的技巧和方法。

工业和信息化人才培养规划教材——高职高专计算机系列

数据结构（第2版）

◆ 编　　宗大华　陈吉人
责任编辑　桑　珊

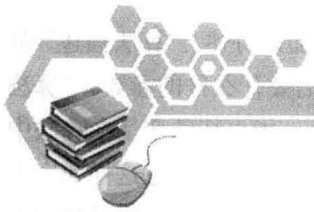
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京铭成印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：17.5　　2013年5月第2版
字数：446千字　　2013年5月北京第1次印刷

ISBN 978-7-115-30807-8

定价：39.80元

读者服务热线：(010)67170985　印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第0021号



第 2 版前言

无论人们要让计算机做什么，都必须涉及三件事：第一，确定要处理的数据之间存在什么关系，以便能够根据这种关系知道一个数据的后面应该是哪一个数据；第二，确定要对数据做哪些处理，是插入、删除、查找，还是排序；第三，确定以何种方式把数据存放到计算机的内存，并反映出它们之间存在的关系，以对它们进行加工处理。

“数据结构”就是研究这些问题的，是从现实世界的实际需要中抽象出来的，在计算机科学各个领域以及系统软件中都有着广泛的应用。当前，“数据结构”是计算机学科的一门重要专业基础课，也有成为许多非计算机专业，特别是工科专业重要技术基础课的趋势。

本书共 9 章，分为三大部分。

第一部分是第 1 章，介绍数据的逻辑结构（即数据间的各种关系）、存储结构（即数据存储在内存中的方式）、算法（即在数据上做的各种处理）和算法的分析，该章是对数据结构的概述，是整本书的基础。

第二部分包括第 2 章到第 7 章的内容，介绍线性表、堆栈、队列、串、数组、矩阵、二叉树、树、图等各种具体的数据结构、存储实现及其常见算法。

第三部分由第 8 章和第 9 章组成。第 8 章介绍查找技术：静态的（折半查找和分块查找）和动态的（二叉查找树和散列表）；第 9 章介绍排序技术：插入排序（直接插入、折半插入、表插入）、交换排序（冒泡、快速）和选择排序（直接选择、堆）。

由于是专业基础课程，因此在教学计划中，数据结构课总是安排得比较靠前。基于数据结构课程本身理论性、抽象性较强的特点，以及当前高职高专学生的认知能力和水平，本书在编写过程中做了如下努力。

1．精心选取内容

本书对各章前后内容的安排做了详尽的设计和考虑。例如，只简略地介绍算法分析的概念，让学生知道应该从什么角度去衡量一个算法的优劣，但对于后面的算法，并没有去讨论它们的性能；在第 2 章和第 3 章中，加入了有关数据结构的一些实际应用，以期扩大学生的视野；把串、数组、矩阵合并成一章，只介绍不多的串运算，删除了有关广义表的内容；考虑到递归实现较为抽象，学生理解困难，因此在算法描述上，侧重于给出非递归的算法，只是偶尔牵扯递归问题；把二叉树与树分成两章来讲述，突出它们之间的区别，分散难点；由于第 8 章是介绍查找的，因此把其他教材中通常称为“二叉排序树”的查找方法，统一称做“二叉查找树”，以免学生发生误解；本书只涉及内排序，把关于外排序的内容全部删去。

2．配备大量例题和习题

在介绍一个概念后，大都会安排例题以帮助学生对概念做出正确理解；在给出一个算法前后，总会配有例子加以解释或说明。这些例子不一定都要在课堂中给学生讲授，完全可以留给他们课下去自己阅读和理解。另外，在每章的最后，都附有大量的习题，以帮助学生自学和检验学习效果。

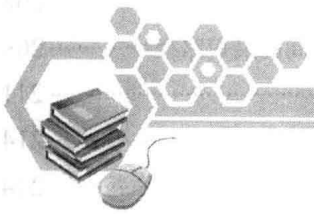
3. 详解每一个算法

对于全书的算法，大都从算法描述、算法分析和算法讨论3个方面加以讲述。“算法描述”就是用类似于C语言的形式给出具体的算法，指出算法的存储结构以及需要什么样的参数；“算法分析”就是对算法的结构、功能做出解释，细说各个部分要完成的任务，指出特殊变量在算法中的意义；“算法讨论”就是讲述有关算法的改进、扩展、注意事项等。总之，希望通过这样3个方面对算法的介绍，能够使学生更好地理解算法，更快地掌握算法，更希望学生能够从中感悟到程序编写的技巧和方法。

书在修订过程中，得到了多位同事、朋友的帮助，在此就不一一详列，仅致以诚挚的谢意。由于编者水平有限，书中必定会存在瑕疵甚至谬误，衷心希望广大读者批评和指正。

编　者

2012年12月



目　录

第1章 数据结构概述

“数据结构”是计算机专业的一门重要基础课程，它研究的问题是从实际需要中抽象出来的，是计算机科学各领域以及系统软件都会用到的知识。例如，语言编译程序的实现要用到栈、散列表、语法树；操作系统要使用队列、存储分配表、目录树，来对整个计算机系统的软、硬件资源（如CPU、存储器、外部设备、文件）实施管理；数据库管理系统工作时，将通过线性表、索引树对数据进行快速搜索查找；而在网络设计技术中，会涉及求解最小生成树、最短路径的问题。这里列举出的内容，如线性表、栈、队列、链表、树、图等，都将是后面具体章节里所要学习的数据结构的内容。

本章是全书的基础，将在此介绍有关数据结构一些最为基本的概念。

本章主要介绍以下几个方面的内容：

- 数据的逻辑结构；
- 数据的存储结构；
- 数据处理算法的描述与分析。

1.1 数据的逻辑结构

计算机是对数据进行处理的工具。具体地说，就是对于一组输入的数据，通过计算机的加工处理，得到相应的输出数据。因此，无论人们要让计算机做什么样的大事、小事、繁杂事、简易事，在用计算机语言编写出程序、执行程序得到处理结果之前，都必须涉及这样的3个问题：第一，确定所要加工处理的数据之间的关系，以便进行处理时，能够知道一个数据的后面是哪一个数据，这种数据间的邻接关系，就是所谓的数据的逻辑结构问题；第二，确定要对数据做哪些处理，是插入、是删除、是查找、还是排序，这就是所谓的算法描述问题；第三，确定以何种方式把数据存放到计算机的内存，并反映出它们之间的邻接关系，从而有利于对它们进行加工处理，这就是所谓的数据的存储

结构问题。

本节首先介绍在现实生活中，很容易抽象出的各种数据的逻辑结构，也就是数据间的邻接关系。

1.1.1 数据及数据间的邻接关系

“数据”是大家非常熟悉的一个词汇，它是信息的载体，是人们用符号来表示客观事物的一种集合。在早先，只要提及数据，人们就认为是指能够参加运算的那些数字（例如大家熟悉的整数和实数）。自20世纪40年代中期计算机问世之后，数据的内涵就有了扩展，计算机处理的数据由传统的数字扩大到了字符串（如英文或中文文本）、逻辑值（“真”和“假”）等。随着计算机科学的进一步发展，随着计算机应用的深入与普及，当前计算机能够处理的数据更是扩大到了表格、图形、图像、色彩、语音等。

因此，现在可以把“数据（Data）”定义为：所有能够输入到计算机中被计算机加工、处理的符号的集合。这就是说，现在说到的数据，已经完全不同于早先人们头脑中理解的“数”的概念了，这是必须首先要明确的事情。

可以把计算机处理的数据，笼统地分成数值型数据和非数值型数据两大类，分别应用于数值计算问题和非数值计算问题。当前，计算机大量地被用于文字、表格、声音、图像等非数值计算领域，并对这样的一些数据进行着各种各样的加工处理。一般地，数值计算问题的特点是加工时的数据量小，处理方法复杂。

非数值计算问题的特点是加工时的数据量大，所处理的数据之间存在着某种特定的关系，计算机在处理它们时，需要对那些关系进行表示。至于数据处理，其方法相对简单得多，但不再局限于单纯的数值计算，而是要对它们进行组织、管理、维护、检索等。

通常，数据是由一个个“数据元素（Data Element）”（简称“元素”）集合而成的。在不同场合，数据元素也常被称作“结点”、“顶点”或“记录”。每个数据元素都具有完整、确定的实际意义，是数据加工处理的对象。例如，表1-1列出的是某公司雇员的信息，每个雇员的信息就是一个元素，由它反映各雇员的实际情况。

表1-1 某公司雇员的信息

雇员号	姓名	年龄	性别	住址
071253	庄严肃	34	男	滨江东路蔷薇新村66号
073825	刘歆子	28	女	春林路北街34号
072154	陈希平	24	男	柳浪街松林坡12栋8号
071546	李汕鸣	30	男	东春斜街甲77号
076671	闻风姗	35	女	锦思门前街2门68号
074533	黄晋民	28	男	解放大桥街66号

一个数据元素又可以细分成由若干个“数据项（Data Item）”组成，数据项也常称作“字段”或“域”。数据项是数据元素中不可再分割的最小标识单位，通常不具备完整、确定的实际意义，只是反映数据元素某一方面的属性。例如雇员信息中的“雇员号”、“姓名”等，都是数据项（或

“字段”、“域”)。

大千世界里，每一个数据的实际意义真是太不一样了。在数据结构中并不去关心数据元素的实际意义，只是把它们抽象地视为一个个被加工的对象。数据结构所关心的，是如何从一个数据能够找到另一个数据的那种数据间的“关系”，人们必须根据那种关系来组织和存储数据，以便顺利、有效地完成对一组数据的各种处理要求。

如果两个数据结点之间有着某种逻辑上的联系，那么就称这两个结点是“邻接的”。若用圆圈代表结点，用结点间的一条连线代表它们之间存在的逻辑关系，那么，就可以用图 1-1 来表示结点 A 和 B 是“邻接的”。在现实世界里，数据间的逻辑关系，可以体现为是前后关系、上下级关系、父子关系、连接关系等。

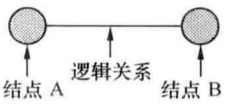


图 1-1　结点的邻接

1.1.2　数据的逻辑结构

从大量实践中抽取出来的、常见的数据间的邻接关系有 3 种：线性关系、树型关系以及图状关系。数据间的邻接关系，就是数据的“逻辑结构（Logical Structure）”。

1．线性关系

所谓数据间具有“线性”关系，是指数据一个接一个地排列成一行。把数据组织成这种线性关系，可能是人们最常见、最熟悉的做法了。上面所给出的公司雇员表，其数据元素之间就是一种线性关系。如果所要处理的数据之间呈线性关系，那么就说它的逻辑结构是线性的。

在线性关系中，把排在第 1 个位置的结点称为起始结点，把排在最后一个位置的结点称为终端结点，其余的结点都称为中间结点，如图 1-2 所示。

图 1-2　线性关系中的各种结点

数据间线性关系的特点是：除起始结点和终端结点外，每个结点的前面有且只有一个结点与它相邻接，每个结点的后面也有且只有一个结点与它相邻接，起始结点的前面没有与之相邻接的结点，终端结点的后面没有与之相邻接的结点。简单地说，数据间线性关系的特点就是：有头有尾，顺序排列。

2．树型关系

所谓数据间具有“树型”关系，是指在数据之间具有分支、层次的逻辑关系。由于这种逻辑关系看上去很像自然界的一棵倒置的树：树根位于最上面，树的叶子在最下面，故而得名为“树型”关系。于是，如果所要处理的数据之间呈树型关系，那么就说它的逻辑结构是树型的。

图 1-3 所示是一个树型结构图例。人们所熟悉的文件目录间的逻辑结构就是树型的。

数据结点间这种分支、层次式组织形式的特点是：第 1 层只有一个结点，它是树型关系的起点；除第 1 层结点和分支末端结点外，位于中间各层的结点的前面只有一个结点与它相邻接，每个结点的后面可以有多个结点与它相邻接；第 1 层结点的前面没有结点与之邻接，每个分支末端结点的后面没有结点与之邻接。

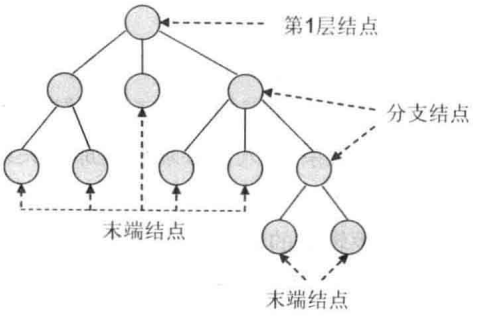


图 1-3　树型结构图例

3．图状关系

如果数据中的任何两个元素之间都可能有邻接关系，那么就说它们之间的关系是图状的。于是，如果所要处理的数据之间呈图状关系，那么就说它的逻辑结构是图状的。不难理解，图状关系是数据间最复杂的关系。

交通网络或通信网络都是图状关系的典型例子。例如，图 1-4 所示为一张航空网络图，图中与结点“武汉”相邻接的结点有“北京”、“广州”、“南京”，而与结点“南京”相邻接的结点有“北京”、“广州”、“武汉”、“上海”。在图状关系中，找不到谁是起点，谁是终点，各个结点的地位可以说都是相同的。

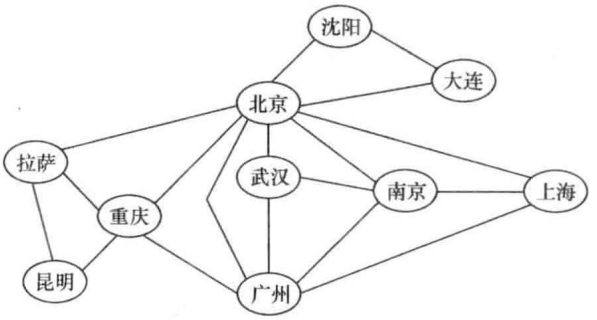


图 1-4　航空网络

图状关系的特点是：每个结点都可能与多个结点有邻接关系。数据间的线性关系和树型关系，都可以视为是图状关系的一个特例。

1.2　数据的存储结构

无论数据间有何种逻辑关系，它要得到计算机的加工处理，就必须存放到内存中才能进行。注意，这里所说的“存放数据”，并不是一个简单的问题，它既要存储数据信息本身，也要存储数据间的邻接关系。因为只有这样，在对数据进行加工处理时，才能够正确、方便、快捷地从一个数据找到与之邻接的另一个数据。

数据的“存储结构（Storage Structure）”，就是研究数据在内存中的存储方式，也就是在内存中有哪些存放数据的方法。数据的存储结构在有些书里也称为数据的“物理结构（Physical Structure）”。从整体上看，数据在存储器内有两种存放的方式：一种是集中地存放在内存中的一

个连续的存储区；另一种是利用存储器中的零星区域，分散地存放在内存的各个地方。

在把数据存储到存储器时，是以数据元素（即数据结点）为单位进行的。分配给单个数据结点的存储区域，称为一个“存储结点”。如前所述，数据存储在存储器里，既要存储数据本身，还要存储数据间的邻接关系，因此，在一个存储结点里，除了要有数据本身的内容外，还要有体现数据间邻接关系的内容。

1.2.1　顺序式存储结构

所谓数据的“顺序式存储”结构，即是为一组数据分配一个连续的存储区，然后按照数据间的邻接关系，相继存放每个数据。因此，使用这种存储结构，在每个存储结点里只存放数据元素本身，而不去存放反映数据邻接关系的信息。也就是说，数据的这种存储结构，是借助存储结点间内含的位置关系，来体现数据元素间的邻接关系的。

例如，图 1-5 左侧所示为一个数据元素所需要的存储尺寸，本书约定将这个尺寸记为 size 字节，图 1-5 右侧所示为在内存里开辟了一个连续的存储区，用来依次存放数据的若干个存储结点。

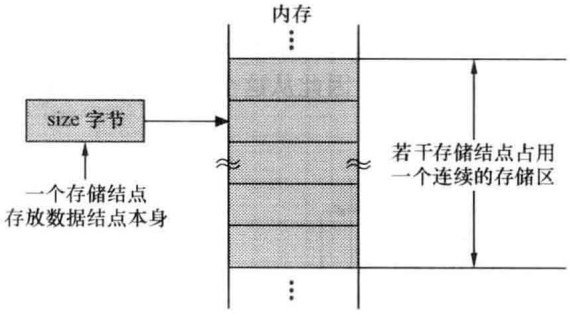


图 1-5　顺序存储结构

假定现在的数据个数为 1000，每个数据元素需要占用 size =16 个字节的存储区。那么，如果采用顺序式存储方式来存放这 1000 个数据，就需要在内存里为它们开辟一个有 1000×size= 16000 字节的连续存储区。

对于顺序式存储结构，要再次强调的是在每一个存储结点里只存放数据，不存放数据间的邻接关系，数据间的邻接关系是借助存储结点本身一个紧接一个存放的位置关系体现出来的。因此，顺序式存储结构对存储的利用率为 100%（即分配给存储结点使用的存储区全部被用来存放数据内容）。可以看出，邻接关系为线性的数据，使用顺序式存储结构最为合适，因为它的存储利用率最高。

1.2.2　链式存储结构

所谓数据的“链式存储”结构，即是存放每个数据的存储结点都由两个部分组成：一部分用来存放数据元素本身的信息，另一部分用来存放与本数据元素邻接的那个数据元素存储结点的位置，即存放指向与之邻接的那个存储结点的指针（起始地址），通过这些指针反映出数据间的逻辑关系。

例如，图 1-6（a）所示为一个链式存储结点，在它的里面除了存放数据元素（用 Data 表示）外，还存放着一个指针（用 Next 表示）。图 1-6（b）表示有 3 个数据元素，分别是数据元素 A、数据元素 B、数据元素 C，它们间的逻辑关系是：数据元素 A 与数据元素 B 邻接，数据元素 B 与数据元素 C 邻接。采用链式存储方式时，存放数据元素 A 的存储结点里，存放着指向数据元素 B 的指针；存放数据元素 B 的存储结点里，存放着指向数据元素 C 的指针；存放数据元素 C 的存储结点里，存放着一个空指针符“∧”，以表示数据邻接关系的结束。

在链式存储结构里，从一个结点的 Next 指针，可以找到它后面的那个结点（即后继）在内存中的位置。因此，必须另设一个指针，指向这种存储结构的第 1 个存储结点。在图 1-6（b）里，用 head 表示这个指针。这样，由 head 就可以找到第 1 个存储结点；由第 1 个结点的 Next 可以找到第 2 个存储结点；如此等等一直下去，直到遇见空指针符“∧”时，表示结束。为了简明起见，常把图 1-6（b）表示成图 1-6（c），用符号“→”表示指向下一个存储结点。

由于链式存储结构是通过指针来体现数据元素之间的逻辑关系的，因此，可以用存储区里的零星小区域来存放数据，而不去动用存储器中的那些大的连续存储区（大的连续存储区可以分配给必须占用连续存储区的数据使用）。从这个意义上来说，链式存储方式提高了存储器的利用率。但另一方面，链式存储结构中的每一个存储结点不仅要存放数据元素自身的信息，还要开辟适当的存储区来存放指针，以反映出数据间的邻接关系。用来存放数据邻接关系的存储区，相对于数据内容来说是额外要求的存储开销。因此从这个意义上说，链式存储结构又降低了存储器的利用率。

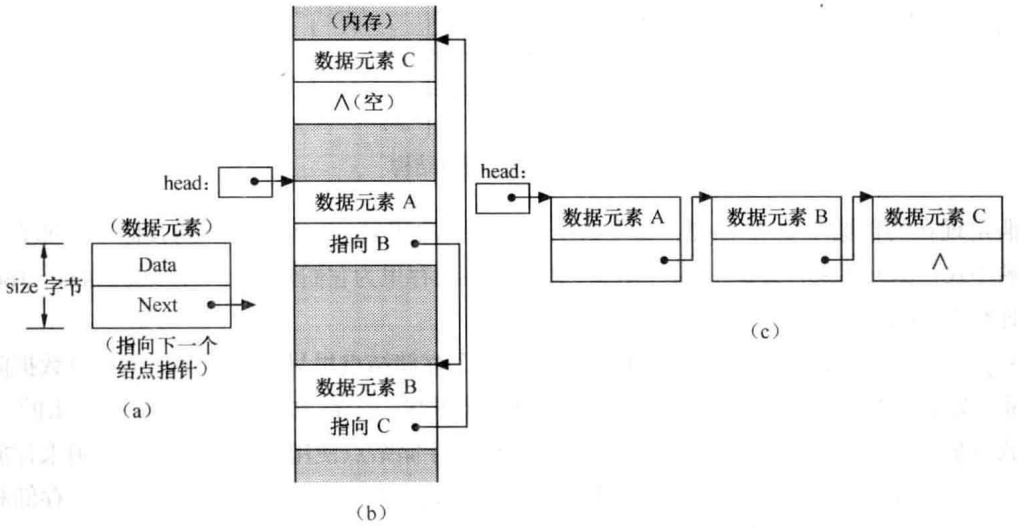


图 1-6　链式存储结构

例如仍假定现在的数据个数为 1000，存储每个数据元素需要占用 16 个字节的存储区，存储每个指针需要占用 2 个字节的存储区。那么，一个存储结点就需要占用 size =18 个字节（这 18 个字节当然必须是一个连续的存储区）。这样，如果采用链式存储结构来存放这 1000 个数据，它们就需要在内存里总共占用 1000×size =18000 字节的存储区才行。这就是说，采用链式存储结构时，要比采用顺序式存储方式多耗费 2000 个字节。

在链式存储结构中，存储结点里的指针并不局限于只能是一个，而是可以根据问题的需要安排为多个。如果采用链式存储结构时，存储结点里只有一个指针，则称是单链式结构；如果存储

结点里有两个指针，则称是双链式结构；如此等等。这些内容，在后续章节会做详细的介绍。

1.3　算法及算法分析

人们关注数据的逻辑结构（即邻接关系）以及数据在内存中的存储结构，最终目的是要使用计算机对数据进行各种加工处理，例如插入、删除、查找、修改、排序等。

为了实现对数据的加工处理，如果问题很简单，那么就可以直接用某种计算机程序设计语言编写程序，然后投入运行得出结果。但是如果问题较大、较复杂，那么这一过程最好分两步完成：先是通过分析列出与加工处理相关的各个步骤，然后再去用某种计算机程序设计语言编写出相应的程序在计算机上运行。这第一步就是所谓的“算法描述”，第二步就是所谓的“程序实现”。

1.3.1　算法及算法的描述

1．算法和程序的区别

人们常把算法和计算机程序等同起来看待，其实它们是两个不同的概念。

所谓“算法（Algorithm）”，是指解决问题的一种方法步骤或者一个过程。对于一个问题，可以用多种算法来解决；而一个给定的算法，则只解决一个特定的问题。例如，本书后面要介绍的数据排序问题，就可以给出很多种算法。当然，每一种算法适用的场合以及性能指标是不一样的。

由于算法是“解决问题的一种方法步骤或者一个过程”，因此一个算法应该具有以下几个重要的特征。

（1）输入：一个算法应该有 n（$n \geq 0$）个初始的输入数据。

（2）输出：一个算法可以没有或有一个或多个输出信息，它们与输入数据之间会有着某种特定的关系。

（3）确定性：算法中的每一个步骤都必须具有确切的含义，不能有二义性。

（4）可行性：算法中描述的每一个操作步骤都必须是可以执行的，也就是说，都可以通过计算机实现。

（5）有穷性：一个算法必须在经历有限个步骤之后正常结束，不能形成死循环。

例 1-1　判断下面用文字描述的计数过程是否构成一个算法。

（1）开始；

（2）n=0；　　/* 变量 n 赋初值 0 */

（3）n=n+1；　　/* 变量 n 增 1 */

（4）重复执行（3）；　　/* 循环执行增 1 操作 */

（5）结束。

解：初看起来，这个计数过程只有 5 个步骤，具有有穷性。但实际上，该过程只要执行起来，就会永远无休止地在变量 n 上面重复做加 1 的操作，形成一个死循环。所以，它并不是一个正确的算法。

例 1-2　编写一个算法，按照从小到大的顺序排列两个数值变量 x、y 的内容，即要求最终有 $x \leqslant y$。

解：用文字描述解决这个问题的算法如下：

（1）输入变量 x、y 的数值；

（2）把两个数值中的小者存放到 x 里；

（3）把两个数值中的大者存放到 y 里；

（4）输出 x、y 的值。

可以看出，上面的描述符合算法的 5 个特征。

所谓“程序（Program）”，是指使用某种计算机程序设计语言对一个算法的具体实现。例如，例 1-2 给出的“按照从小到大的顺序排列两个数值变量 x、y 的内容”的算法，可以用如下的 C 语言程序来实现。

```
#include "stdio"
main()
{
  int x, y, temp;
  scanf ("%d%d", &x, &y);                    /* 从键盘输入两个整型数据 */
  if (x>y)                                   /* 对数据进行比较 */
     {temp = x ; x = y ; y = temp ; }
printf ("x = %d, y = %d\n", x, y);           /* 打印输出 */
}
```

对比算法和程序，可以看出算法侧重于对解决问题的方法描述，即要做些什么；而程序是算法在计算机程序设计语言中的实现，即具体要怎样去做。例如，例 1-2 中只是讲“把两个数值中的小者存放到 x 里”，讲“把两个数值中的大者存放到 y 里”，并不去管到底怎样去比较和存放。但是，在例 1-2 的 C 语言程序实现中，则要给出具体的比较和存放方法，即

```
if (x>y)
   {temp = x ; x = y ; y = temp ; }
```

可见，严格地讲算法与程序是两个不同的概念。

当然，也可以直接把计算机程序看作是对解决问题方法的一种描述，那么算法和程序就是一回事了。本书为了简化表述的过程，加之学习数据结构课程之前都已学习过 C 语言程序设计，因此不去过分地强调“算法”和“程序”的区别。

2. 算法的描述

算法是可以用不同方法来描述的，下面给出几种常见的方法。

- 算法描述方法 1：使用人们习惯的自然语言来描述算法。

上面的例 1-2 就是用自然语言描述的一个算法。下面的例 1-3 采用的也是这种方法。

例 1-3 用自然语言描述输出整数 1、2、3、…、9、10 的过程。

解：用自然语言描述输出整数 1、2、3、…、9、10 的过程的算法如下：

（1）开始；

（2）将初始值 1 赋予变量 i；

（3）如果 i>10，则转向（7）；

（4）输出 i 的值；

（5）将 i 的值加 1，再赋予 i；

（6）转向执行（3）；

（7）结束。

- 算法描述方法 2：使用人们熟悉的流程图（即框图）来描述算法。

所谓“流程图”，即是利用不同形状的图形以及一些带箭头的线条，来描述算法中的各个操作步骤。图 1-7 给出了流程图中可能出现的各种图形的名称和作用。

下面给出的例 1-4 采用的就是用流程图描述算法的方法。

例 1-4　用流程图描述输出整数 1、2、3、…、9、10 的过程。

解：用流程图描述输出整数 1、2、3、…、9、10 的过程的算法如图 1-8 所示。图中的两个圆角矩形框分别表示算法的开始和结束；两个矩形框分别表示要做的操作；一个菱形框表示条件判断；一个平行四边形框表示输出。

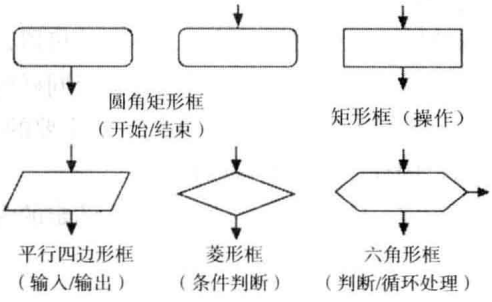


图 1-7　流程图的各种图形名称和作用

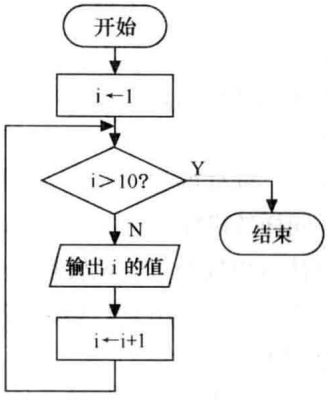


图 1-8　用流程图描述算法

- 算法描述方法 3：用“类 C 语言”来描述算法。

所谓“类 C 语言”，即是对 C 程序设计语言做一些简化，略去对算法描述来说是无关紧要的部分（例如变量说明）。采用类 C 语言来描述算法的好处是，既可以利用 C 语言强大的描述功能，又能使人们不用去拘泥于 C 语言繁杂的语法规则。用“类 C 语言”描述的算法不能直接在计算机上执行，但人们很容易将它改写成 C 语言程序。

本书将采用类 C 语言来描述算法，这样读者可以不必把精力放在 C 语言及其结构上，而是主要去关注算法实现的基本思想。

- 算法描述方法 4：直接采用 C 语言来描述算法。

例 1-5　分别用 C 语言和类 C 语言来描述输出整数 1、2、3、…、9、10 的过程。

解：（1）用 C 语言描述输出整数 1、2、3、…、9、10 的过程的算法如下。

```
void num ()
{
  int i;
  i=1;
  while (i<= 10)
  {
    printf ("i = %d\n", i );
    i = i +1;
  }
}
```

（2）用类C语言描述输出整数1、2、3、…、9、10的过程的算法如下。

```
void num ()
{
  i=1;
  while (i<= 10)
  {
    printf ("i = %d\n", i );
    i = i +1;
  }
}
```

由例1-5可以看出，程序与算法很相似，但二者之间是有一定差异的。最大的差异表现为算法与机器无关，它不依赖于某个机器，不依赖于某种语言（所谓用“类C语言”来描述算法，并不是说算法描述就依赖于C语言，只表明它的语法与C语言的语法类似。有些书用“类Pascal语言”来描述算法，那就表示这种描述语言的语法与Pascal语言的语法类似）。

程序是可以执行的，因此对于机器具有一定的依赖性：同样的加工处理，不同的机器，程序是不同的；同样的加工处理，同样的机器，不同的程序语言，程序也是不同的；甚至同样的加工处理，同样的机器，同样的程序语言，由于版本不同，其程序也会有差别。不过，在需要的时候，一个算法可以较为方便地转换成为特定机器、特定语言的程序，从而得以执行。

下面是本书采用类C语言描述算法时遵守的一些约定，只要熟悉C语言，这些约定的含义都是不难理解的。

（1）尽量不对算法中涉及的变量做具体说明

（2）赋值语句

<变量名>=<表达式>;

（3）所有算法都以函数的形式给出，即

```
<函数类型> <函数名>(<函数参数表>)
{
  <类C语句>
}
```

除非有必要，否则一般不对<函数参数表>里使用的变量做说明。

（4）分支语句

条件语句：if(<条件>)<语句1>;[else<语句2>;]

多分支语句：switch(<表达式>)

```
{
  case 常量值1：<语句1>; break;
  case 常量值2：<语句2>; break;
       ……          ……
  case 常量值n：<语句n>; break;
  default：<语句n+1>;
}
```

（5）循环语句

while循环语句：while(<条件>)<语句>;