21 21世纪全国高等院校计算机教材

数据结构教程与题解

（用C/C++描述）

胡圣荣　周霭如　罗穗萍　编著

北京大学出版社
http:// cbs.pku.edu.cn

TP311.12
61

数据结构教程与题解
（用C/C++描述）

胡圣荣　周霭如　罗穗萍　编著

北京大学出版社

·北　京·

内 容 简 介

本书介绍了线性表、栈、队列、串、多维数组、广义表、树、图、查找表、排序表等多种常用数据结构的数据表示和数据处理方法，包括逻辑结构、存储结构、基本运算及相应的算法，其中算法描述采用了基本的 C/C++语言。

本书力求通俗易懂，概念明确，课后练习和参考答案可作为正文的补充。

本书可作为计算机和信息类相关专业的本（专）科“数据结构”课程的教材和参考书。

图书在版编目(CIP)数据

数据结构教程与解题：用 C/C++描述/胡圣荣等编. —北京：北京大学出版社，2003.8

ISBN 7-301-06472-1

Ⅰ. 数… Ⅱ. 胡… Ⅲ. ①数据结构—高等学校—教材 ②数据结构—高等学校—解题 ③C 语言—程序设计—高等学校—教材 ④C 语言—程序设计—高等学校—解题 Ⅳ. ①TP311.12 ②TP312

中国版本图书馆 CIP 数据核字(2003)第 070235 号

书　　名：数据结构教程与题解（用 C/C++描述）
著作责任者：胡圣荣　周霭如　罗穗萍　编著
责 任 编 辑：黄庆生　汉　明
标 准 书 号：ISBN 7-301-06472-1/TP・0730
出　版　者：北京大学出版社
地　　址：北京市海淀区中关村北京大学校内　100871
网　　址：http://cbs.pku.edu.cn，http://www.macrowin.net
电　　话：发行部 62750672　62765127　编辑部 62765126　邮购部 62752015
电 子 信 箱：macrowin@macrowin.net，xxjs@pup.pku.edu.cn
排　版　者：北京东方人华北大彩印中心　　电话：62754190
印　刷　者：河北涿县鑫华书刊印刷厂
发　行　者：北京大学出版社
经　销　者：新华书店
787 毫米×1092 毫米　16 开本　18 印张　432 千字
2003 年 8 月第 1 版　2003 年 8 月第 1 次印刷
定　　价：25.00 元

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第1章 概 论

随着计算机的普及和软硬件技术的发展，计算机的应用越来越广泛。但不管计算机作何用途，每一项应用总是某个程序的运行。所以，用计算机解决任何问题都离不开程序设计，而程序设计的实质就是数据的表示和数据的处理，数据结构就是研究这两个方面的一些基本问题的，包括如何组织数据、数据元素之间是什么关系、数据在计算机中如何表示以及如何对数据进行操作等。数据结构对设计高性能程序和软件至关重要。

本章介绍了数据结构的基本概念，包括数据的逻辑结构、存储结构、基本运算和运算的实现以及算法分析等。

1.1 引 言

在现实生活中，当我们谈到事物的“结构”时，一般是指它由哪些部分组成，各部分之间的相互关系如何等，如对于计算机课程的体系结构，我们会关心它有哪些课程、各课程之间的关系如何等。所以，对“数据结构”这个概念，从字面上可以理解为数据的组成和相互间的关系，或称数据的组织形式。不过这并不全面，因为数据结构中还应包含数据的相关操作。本章后面会逐步对数据结构进行解释，其中会涉及到很多相关概念。这里先看另一个问题：我们为什么要学习数据结构？或者说学习数据结构有什么用？

简单地说，学习数据结构是我们编程的需要。这是因为，不论什么程序，它本质上都是计算机对某种数据的加工处理，计算机相当于一个处理机，数据是它加工处理的“原料”。这里涉及两个基本问题：首先，数据要存储到计算机中，才能被计算机加工处理；其次，如何对数据进行处理。前一个问题称为**数据表示**，后一个问题称为**数据处理**。所以，程序设计的实质就是数据的表示和数据的处理。

数据在计算机存储器内的存在形式称为机内表示，这之前的数据表现形式称为机外表示，所以数据表示的工作就是将数据从机外表示转化为机内表示。在数据表示中，不是简单地将数据的值存储到计算机中就可以了，还要直接或间接地存储数据之间的相互关系，对关系的表示常常是一些复杂问题的关键。数据处理的工作就是用计算机可执行语句编制程序，描述对已存入计算机的数据进行各种具体操作，继而完成整个处理任务。数据结构就是研究程序设计过程中数据表示和数据处理这两个方面的一些基本问题的。

或许有读者问：我们以前学习高级语言如 C/C++语言时，并没有学过数据结构，不是一样也编出了很多程序并且运行得很好吗？这里有一个认识问题。首先，我们在学习高级语言时所编制的程序基本上是属于数值计算型的，如级数求和、方程求根等，所涉及的运算对象比较简单，如整数、实数等，数据结构的问题不明显；其次，即便如此，我们在编程中也不知不觉地、或多或少地使用了数据结构的一些知识或方法。最后，从数据结构的

观点看，即使是一个单独的数据，如一个整数或字符，也可看成一个简单的数据结构。

下面先看一个简单的例子。

例 1.1 方程求根

$f(x)=a_2x^2+a_1x+a_0=0$

显然，首先要把方程的系数存入计算机，如用 3 个变量 a0、a1、a2 表示，然后才能进行具体的求根计算。这就是我们在学习高级语言时一般采用的方法。但对一般形式的方程：

$f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0=0$

我们就一般不会对每个系数都用单独的变量来表示了，因为变量多，对名字的管理和书写都很不方便，如 20 次的多项式，就需要 21 个变量名。这时，我们一般会用一个数组如 A[21]来集中存放这些系数，这样只需记住一个名字，即数组名，然后通过数组下标来区分各个系数，如 A[i]表示 a_i。这里，数组就是数据结构中一种比较简单的存储结构——顺序存储结构，它将各个元素在存储空间上连续存放。

如果多项式的次数不定，上述方法又有问题。如 C/C++等语言中数组的大小一经定义后就不能再变化，这时，如果数组定义得较大，而多项式的次数低，实际系数少，数组空间就有浪费现象；反之，如果数组定义得较小，而多项式的次数高，实际系数多，数组空间就可能不足而产生溢出。为适应数据个数变化的情况，我们可用内存分配函数为每个系数动态分配空间，并通过指针将它们联系起来，即得到链表。这里，链表也是数据结构中常用的一种存储结构——链式存储结构。

如果再考虑到多项式的系数中通常有很多为零，为了节省存储空间，并和书写习惯一致，我们可不存储零系数，但这时要把非零系数和原多项式之间的关系表示清楚，这就涉及数据的压缩存储问题，也是数据结构里要研究的内容。

在上面存放系数时，不论数组还是链表，我们显然不会胡乱存储，而是很自然地按照系数间的“前后”次序，即按对应项的次数从高到低或从低到高来进行。这实际上说明了这样一个问题：这些系数间是有相互关系的，对本问题，它们按次数的高低顺序形成一个有穷序列$(a_0,a_1,a_2,\cdots,a_n)$，对任一个系数 a_i，排在它前面的与之相邻的系数以及排在它后面的与之相邻的系数都最多只有一个，这是数据结构里一种比较简单的逻辑结构——线性结构。

将系数存储到计算机中后，就可以进行求根计算了。对低次方程如 2 次、3 次，可用求根公式，但一般情况下要采用迭代法，如牛顿迭代法。在计算中，要涉及 x 的各次方 x^i，显然不必每个都单独计算，否则会有很多重复计算，如 x^3 不必用 x*x*x 来计算，可以在已有 x^2 的基础上再做一次乘法 $x*x^2$ 即可。这里，怎样实现具体的计算方法，属于算法问题，而怎样评价算法的效率，就是算法分析问题。这都是数据结构里要讨论的内容。

可见，对这个例题，我们已经“无意间”用到了数据结构的一些相关知识，同时也看到，即使问题不太复杂，如果只有程序语言方面的知识，将数据如何有效地组织起来，以及如何对数据进行有效的运算已显得有些“力不从心”了。

上述例子属于数值计算问题，在计算机发展初期，人们使用计算机主要就是处理这类问题的。由于所涉及的数据对象比较简单，如整型、实型或布尔型等，数据结构的问题不

突出，程序设计者的主要精力集中在程序设计的技巧上，解决此类问题的关键是数值计算方法（算法），程序以算法为中心。

但是，大量的实际问题是仅凭高级语言的知识无法处理的，必须主动地借助于数据结构的知识。这是因为，随着计算机软硬件的发展和计算机的普及，计算机应用领域不断扩大，早已不局限于科学计算了，大量的应用，如文字处理、数据库、多媒体、游戏、过程控制等都是非数值型问题。在这些问题中，要处理的数据一般比较复杂，如字符、表格、图像、声音等，这不仅是数据的内容比较复杂，更主要的是数据之间还有复杂的关系，而这些关系无法用数学方程式描述；另外，对数据的处理一般也很复杂。于是，如何有效地组织数据以及如何对数据进行有效的运算等问题就成了这类问题的关键，而这正是数据结构所要研究的。

例 1.2　用计算机进行部门机构管理。

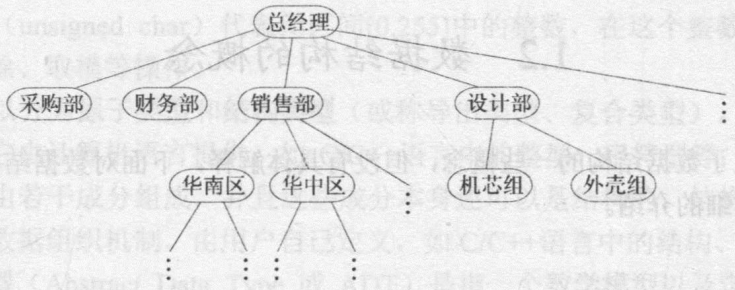


图 1.1　机构组成示意图

这里，要处理的数据是整个机构，它由各个部门组成。显然要将各个部门的信息存储到计算机中，如员工数、位置、编号、名称等，我们可用结构（或称记录）类型的量来表示这些信息。然而，对本问题，理顺部门之间的关系，或者说对关系的表示才是问题的关键：各部门间可能是上下级关系，也可能是同等关系，整个部门机构组成一种层次结构。如果不把这些关系表示出来，这个问题根本就没法解决。层次关系可很自然地用图 1.1 的形式表示，其中，每个部门用一个椭圆表示，椭圆间的连线表示上下级关系：每个部门最多只有一个直接上级，但可有多个直接下级。数据的这种层次结构在数据结构里称为树。

所以，在存储部门数据时，还要将部门间的关系存储起来或反映出来。我们可以将所有部门按某种次序依次存放到一片连续的内存单元中，并在每个部门中增加上下级关系信息，比如增加上级关系信息，这可用一个指针来表示，即指出它上级部门的存储位置。这样，从任一个部门通过指向上级的指针可找到它的所有上级；如果某个部门的上级指针为空，则它是最上级的部门；如果要考察某两个部门之间有没有上下级关系，则只要看其中某个部门的上级中有没有另一个就可以了。这是树形结构的一种双亲存储结构。

将部门数据存储到计算机中以后，就可对其进行有关的管理工作了，如新部门的增加（插入）、旧部门的撤销（删除）、部门的查询等，这些操作的具体实现过程就是本问题的算法。

上面两个例子用到的数据结构是线性表和树。一般说来，为了有效地表示数据，特别是数据之间的关系，需要设计和采用合适的数据结构，在此基础上才能写出有效的算法完成数据的处理，从而有效地解决实际问题。在实践中经常会遇到这种现象：同一个问题，

采用一个“好”的数据结构很快就能完成运算，而采用一个“差”的数据结构，运算时间可能会相差几倍甚至几十倍或更多，比如研究一下计算机图形学中的一些消隐算法就会深有体会。

著名的瑞士计算机科学家沃思（N.Wirth）教授曾提出：算法＋数据结构＝程序。这里的数据结构是指数据的逻辑结构和存储结构，而算法则是对数据运算的描述。由此可见，程序设计的实质也可看成是对实际问题选择一种好的数据结构，加之设计一个好的算法，而好的算法在很大程度上取决于描述实际问题的数据结构。

数据结构是计算机软件和计算机应用专业的一门核心课程，也是一些计算机软件相关专业的重要课程，在众多的计算机系统软件和应用软件中，都要用到各种数据结构。仅靠掌握几种计算机语言是难以应付众多复杂问题的，要想有效地使用计算机解决实际问题，必须积极主动地学习和应用数据结构的有关知识。

1.2 数据结构的概念

上面已涉及了数据结构的一些概念，但没有具体解释，下面对数据结构的一些概念和术语进行比较详细的介绍。

1.2.1 数据

从数据结构的观点看，通常所说的“数据”应分成三个不同的层次，即数据、数据元素和数据项。

数据（Data）：凡能被计算机存储、加工处理的对象通称为数据。它是计算机程序加工处理的对象和原料。前已指出，早期的计算机主要用于科学计算，数据的概念主要是指整型、实型或布尔型等数值型数据；随着计算机软硬件的发展和计算机的普及，计算机应用领域不断扩大，数据的概念已逐步扩展到字符串、表格、图像甚至语言等。

这里随便提一下与数据密切相关的另一个概念：信息。简单地说，信息是加工处理后的数据，是数据的内涵；而数据是信息的载体，信息需要通过数据表示出来。如 A、B 两个人成绩分别是 85 和 95 分，我们得到的信息是他们的成绩都较好，但 B 更好。反之，为了说明一个人的成绩好，我们需要给出具体数据，如 90 分（或与之相关的量）。但有时数据和信息并不严格区分，如数据处理也称信息处理。

数据元素（Data Element）：数据的基本单位，在程序中作为一个整体加以考虑和处理，通常具有完整确定的实际意义。有些情况下，数据元素也称为元素、结点、顶点或记录。

数据项（Data Item）：数据不可分割的最小标识单位，具有独立含义，但通常不具有完整确定的实际意义，或不被当作一个整体看待。有时数据项也称为字段或域。数据元素一般由若干个数据项组成，但有时也可只含有一个数据项。

数据、数据元素和数据项反映了数据组织的三个层次，数据可由若干数据元素组成，数据元素又可由若干数据项组成。例如，对例 1.2 的部门机构组织问题，数据即指所有部

此为试读，需要完整PDF请访问：www.ertongbook.com

门构成的整体；其中每个部门就是一个数据元素，因为在此问题中它被当作运算的基本单位。如删除、插入等运算作用的对象就是某个部门，不是整个机构，也不是某个部门中的个别项目。部门的名称、编号等项目则为数据项，它们只表示部门某一方面的信息，在本问题中单独存在时没有完整确定的实际意义。

1.2.2　数据类型

数据类型是与数据结构密切相关的一个概念。它最早出现在高级程序设计语言中，用以刻划程序中操作对象的特性。在用高级语言编写的程序中，每个变量、常量或表达式都有一个确定的数据类型。

数据类型（Data Type）是具有相同性质的计算机数据的集合及在这个数据集合上的一组操作的总称，它显式或隐式地规定了数据的取值范围和操作特性。例如，C/C++语言中的无符号字符型（unsigned char）代表闭区间[0,255]中的整数，在这个整数集中可以进行加、减、乘、整除、取模等操作。

数据类型可以分为**原子类型**和**结构类型**（或称导出类型、复合类型）。原子类型的值是不可分解的，它由计算机语言提供，如 C/C++语言中的整型、字符型等；结构类型的值是可分解的，即由若干成分组成，并且这些成分本身还可以是结构的。结构类型要借用计算机语言提供的数据组织机制，由用户自己定义，如 C/C++语言中的结构、数组等。

抽象数据类型（Abstract Data Type 或 ADT）是指一个数学模型以及定义在该模型上的一组操作的总称。“抽象”的含义是指其逻辑特征与具体的软硬件实现（即计算机内部的表示和实现）无关，在用户看来，无论怎样实现，只要其数学特征不变，就不影响其外部使用。

抽象数据类型可以看作数据的逻辑结构和定义在其上的一组操作组成，而逻辑结构又包括数据元素及其相互关系，因此，抽象数据类型一般可以由数据元素、关系及操作三种要素来定义。

抽象数据类型和数据类型实质上是一个概念。例如，各种计算机都拥有的整数类型就是一个抽象数据类型，在用户看来其数学特征相同，而实际上它们在不同处理器上的实现是可以不同的。另一方面，抽象数据类型的范畴更广，它不局限于在各种处理器中已定义并实现的数据类型，还包括用户自己定义的数据类型。

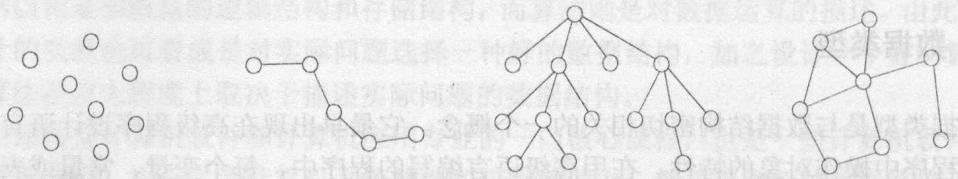
在定义抽象数据类型时，将一组数据和施加于这些数据上的一组操作封装在一起，用户程序只能通过在 ADT 里定义的某些操作来访问其中的数据，从而实现了信息的隐藏。在这个过程中，数据的表示及其操作的细节在模块的内部给出，在模块的外部使用的只是独立于具体实现的抽象的数据及抽象的操作。所以，抽象数据类型的特征是使用与实现相分离，实行封装和信息隐藏。

1.2.3　逻辑结构

为了表示数据间的关系，需要引入逻辑结构的概念。

数据元素对应着客观世界中的实体，数据元素之间必然存在着各种各样的关系，这种数据元素之间的关系就称为**结构**。其中，数据元素之间的关联方式（或称邻接关系）称作

数据的**逻辑关系**，数据元素之间逻辑关系的整体称为**逻辑结构**（Logical Structure）。为了讨论方便，数据的逻辑结构一般可用示意图表示。具体方法为，用小圆圈代表数据元素，用小圆圈之间的连线代表数据元素间的关系，如果强调关系的方向性，可用带箭头的线段表示关系。



(a) 集合　(b) 线性结构　(c) 树形结构　(d) 图状结构

图 1.2　四种基本逻辑结构示意图

有四类基本的逻辑结构，见图 1.2，它们是：

集合：任何两点之间不考虑邻接关系或没有邻接关系，或称作没有关系的关系，其数据组织形式松散，元素之间是“平等”的，它们的共同关系是“属于同一个集合”。也可以说，集合中元素之间除了“同属于一个集合”的关系外，别无其他关系。

线性结构：有且仅有一个开始结点和一个终端结点，并且任何结点都最多只有一个直接前趋和一个直接后继。某点的**直接前趋**（Immediate Predecessor）是指与之相邻且在它前面的结点，**直接后继**（Immediate Successor）是指与之相邻且在其后的结点。开始结点没有前趋，终端结点没有后继。线性结构中数据元素之间存在一个对一个的关系。

树形结构：除了一个特殊元素（根）外，每个元素都只有一个直接前趋，但可有多个直接后继，结点之间具有分支、层次特性。树形结构中数据元素之间存在一个对多个的关系。

图状结构：任何两点之间都可能邻接，结点之间形成网状结构。任一元素都可有多个直接前趋和多个直接后继，元素之间存在多个对多个的关系。

线性结构是一种最常见的数据结构，本书第 2 章、第 3 章介绍的线性表、栈、队列、串等均为线性结构。树形结构与图状结构也称为**非线性结构**，它的逻辑特征是一个结点可能有多个直接前趋或多个直接后继。集合比较特殊，可把它归到非线性结构，因为“线性”之外都是“非线性”，但在实际使用时，也经常对其数据元素增加某种“线性”关系，如出现的先后次序等，按线性结构处理。

有一些数据结构，如多维数组和广义表，尽管本质上属于图状结构，但由于自身的具体特点，与图状结构的处理方法有很大不同，所以一般单独讨论。

关于逻辑结构，有几点需要特别注意：

（1）逻辑结构与数据本身的形式、内容无关。如表示机构组成的树形结构，即使其中某个部门的名称改了，或增加了人员，这个结构仍然是树形结构。

（2）逻辑结构与数据元素的相对位置无关。如将线性结构的各结点按其内容递增或递减的顺序重新排列，则相邻结点之间仍是一对一的关系，即仍是线性结构。

（3）逻辑结构与所含结点的个数无关。如机构组成中，增加或删除几个部门，其结果仍是树形结构。

由此可见，一些表面上很不相同的数据可以有相同的逻辑结构，因此，逻辑结构是数

据组织的某种“本质性”的东西。事实上，逻辑结构是数据组织的主要方面。

在不致于混淆的情况下，本书以后常常将直接前趋简称为前趋、直接后继简称为后继。

1.2.4　存储结构

为了让计算机对数据进行处理，.我们还要研究如何在计算机中表示数据。数据的存储实现（机内表示）称为数据的**存储结构**（Storage Structure）或物理结构，它是指数据元素及其关系在计算机存储器内的表示。一个存储结构一般应包括 3 个内容：

（1）**内容存储**：存储各数据元素的内容（值），每个数据元素占据独立的可访问的存储区。

（2）**关系存储**：直接或间接地（显式或隐式地）存储各数据元素间的逻辑关系。

（3）**附加存储**：一般是为便于运算实现而设置的辅助结点。

其中，前两个部分是所有存储结构都必须具备的，第三个部分则根据实际需要决定是否设置。由于数据元素内部的组织形式一般比较简单，内容存储也就比较简单，所以，逻辑关系的表示是存储结构的主要内容。数据元素存储后，元素间的逻辑关系便由存储结点间的关联方式间接表示。

这里要指出，数据的存储结构是逻辑结构用计算机语言的实现，它依赖于计算机语言和物理设备。但对计算机语言来说，存储结构是具体的，所以我们一般不用具体的物理存储地址来描述存储结构，而只在高级语言的层次上讨论存储结构。数据的存储结构可用以下 4 种基本的存储方法得到：

（1）**顺序存储方式**：所有结点相继存放到一片连续的存储区中，元素之间的逻辑关系通过物理位置关系间接表示。由此得到的存储表示称为顺序存储结构（Sequential Storage Structure），它是一种最基本的存储方法，通常借助程序设计语言中的数组来描述。该方法主要用于线性结构，非线性结构需要通过某种线性化的方法来实现。

（2）**链接存储方式**：结点之间的物理位置不一定连续，它们之间的逻辑关系通过附加的指针来表示。即每个元素的存储区分两大部分：一部分为数据区，存储元素本身的数据内容；一部分为指针区，存储与其他元素之间的关系信息（一般为相关的其他元素的地址）。由此得到存储表示称为链式存储结构（Linked Storage Structure），它通常借助于程序设计语言中的指针来描述，适合存储复杂的数据结构。

（3）**索引存储方式**：在存储结点信息的同时，还建立附加的索引表。索引表中的每一项称为索引项，索引项的一般形式是：（关键字，地址），其中关键字是能标识一个结点的那些数据项。若每个结点在索引表中都有一个索引项，则该索引表称为**稠密索引**（Dense Index），此时索引项地址指出该结点所在的存储位置；若一组结点在索引表中只对应一个索引项，则该索引表称为**稀疏索引**（Sparse Index），此时索引项的地址指示一组结点的起始存储位置。索引存储并不强调关系的存储，而是针对数据内容的，主要面向检索（查找）操作。

（4）**散列存储方式**：以结点的关键字值为自变量，通过某个函数计算出该结点的存储位置（或位置区间端点）。这个函数称为散列函数。散列存储也是面向内容存储的。

实际上，在这 4 种存储方法中，顺序存储方式和链接存储方式是最基本的，因为索引

存储方式和散列存储方式在具体实现时需要利用前两种结构，也可看成前两种结构的衍生。

上述四种存储方法，既可以单独使用，也可以组合使用。有时同一种逻辑结构可采用不同的存储结构，如何选择要视具体要求而定，主要是考虑运算的方便性及算法的时空要求。

1.2.5 运算

为了完成数据处理任务，需要引入运算的概念。

一般地，**运算**是指在逻辑结构上施加的操作，即对逻辑结构的加工。运算与逻辑结构紧密相连，每种逻辑结构都有一个运算的集合。运算的种类很多，随不同的应用而不同。根据操作的结果，运算可分为两种类型：

（1）加工型运算，其操作改变了原逻辑结构的“值”，如结点个数、结点内容等。

（2）引用型运算，其操作不改变原逻辑结构，只从中提取某些信息。

例如，在例 1.2 的树状结构 S 上，一般可定义以下运算：

查找：引用型运算，在 S 中寻找满足一定条件的结点；

插入：加工型运算，在 S 的指定位置上添加新的结点；

删除：加工型运算，删去 S 的某个指定结点；

读取 Get：引用型运算，读取 S 中某指定位置上结点的内容；

更新 Set：加工型运算，修改 S 中某指定结点的内容；

遍历：引用型运算，按某种方式访问 S 中各结点，使得每个结点恰好被访问一次。

关系访问：引用型运算，访问 S 中特定关系的结点，如求某结点的上下级、同级结点等。

根据实际需要，可对这些运算进行增减。

在各种运算中，如果某些运算，它的实现不能利用其他运算，而其他运算可以或需要利用该运算，则这些运算称为**基本运算**。如上面的更新运算就不是基本运算，因为在更新时，可利用查找运算，找到该该点后再修改有关内容；而查找运算是基本运算，它不能利用其他运算。每种逻辑结构都有自己的基本运算集，逻辑结构不同，基本运算集一般也不同。

一般地，我们将较复杂的运算分解为若干较简单的运算，有利于降低程序设计的难度，同时也有利于提高程序设计的效率。在简单运算中，再分解出一些基本运算，当基本运算实现后，其他运算就可通过调用基本运算来实现，进而完成整个程序。

1.2.6 算法

数据的运算是定义在逻辑结构上的，只指出“做什么”，而不考虑“怎么做”。运算实现的这个细节问题就是算法。算法是与数据结构密切相关的一个概念，讨论某种数据结构必然会涉及到相应的算法，而设计某个算法也必然要涉及具体的数据结构。

所谓**算法**（Algorithm），通俗地讲，就是解决特定问题的方法和步骤；较严格地说，就是规则的有穷集合，这些规则规定了一个指令的有限序列，其中每条指令表示一个或多个操作。一个算法必须满足下述准则：

（1）输入：具有零个或多个输入，它们是算法开始前的初始量。

（2）输出：至少产生一个输出，它们是与输入有某种关系的量，是算法的执行结果。

（3）有穷性：算法的执行步骤（或每条指令的执行次数）必须是有限的，整个算法必须在有限步（或有限条指令）后结束。

（4）确定性：算法每一步（或每条指令）的含义都必须明确，无二义性。

（5）可行性：算法每一步（或每条指令）是可执行的，并且执行时间是有限的，整个算法必须在有限时间内完成。

这里要注意，本课程所指的算法是针对数据结构的，而一般问题的算法是面向应用的，它涉及到数据结构的应用，但范围比数据结构中的运算要广。

另外，算法与程序的含义很相似，但二者是有区别的：

（1）程序不一定满足有穷性，即不一定是算法。如操作系统就不是一个算法，因为只要不遭破坏，它就永远不会停止，即使没有作业要处理，它仍处于等待循环中（不过操作系统内部每个具体任务的实现都应是一个算法）。一个程序如果对任何输入都不会陷入无限循环，就是一个算法。

（2）程序中的指令必须是机器可执行的，而算法中的指令虽要求可执行，但不一定是“机器可执行”。如果一个算法用机器可执行的语言来书写，则它就是一个程序，即该算法在计算机上的特定实现。

任何算法都必须用某种方法**描述**出来，其中常用的就是用语言描述。根据描述语言的不同，一般可将算法分为以下三类：

（1）运行终止的程序可执行部分。采用计算机程序设计语言描述，可直接在计算机上运行，从而使给定问题在有限时间内被机械地求解。这类算法比较严谨，但要熟悉计算机语言，有一定难度，也不太直观，常常需要通过注释来提高可读性。

（2）伪语言算法。采用伪程序设计语言描述，不能直接在计算机上运行。伪语言介于程序设计语言和自然语言之间，它忽略程序设计语言中一些严格的语法规则和细节描述，因此伪语言描述可突出算法设计的主要方面而不是语法细节，又比自然语言更接近程序。伪语言算法一般比较简洁，便于编写和阅读，适合于教学和交流，同时也容易修改成程序。

（3）非形式算法。采用自然语言，同时还可使用程序设计语言或伪程序设计语言（如流程控制语句 while、for、if 等）描述。这类算法简单易懂，但不够严谨。

算法除了用语言描述外，实际上还有一种图形描述方法，如流程图、N-S 图等，这种描述更加简单明了。但我们只把它看作语言描述的一个辅助手段，并且它最终还是要用语言描述出来。

不管算法用什么方法描述，它最终都要转换为程序才能在计算机上运行。容易看出上述几种描述方法的可读性依次增强，但可读性越强，离最终程序的距离越远。对一些较复杂问题，一次性地写出它的程序比较困难，一般是先写出伪语言算法或非形式算法，再通过“逐步求精”的过程转化为实际程序。逐步求精的过程也符合人们认识事物的思维活动。

原则上说，任何算法都可以用任一种程序设计语言来实现，但显然具体实现的难易程度和效果会有所不同。随着面向对象程序设计语言的流行，数据结构中越来越多地出现了这类语言的描述，如 C++描述等。应该说，为了较好地描述数据结构，特别是抽象数据类型，以及较好地解决代码重用等问题，这样做是有利的。但这同时也对读者提出了更高的要求，他必须较好地掌握了面向对象的程序设计知识，否则可能出现数据结构的内容不突

出，面向对象的内容倒成了问题的重点和难点。另外，类的构造、析构、继承、重载、多态、模板等面向对象的概念和方法引入后，数据结构的内容常常显得复杂和“高深”起来，在使用中很多初学者都感到比较困难。

为了突出数据结构本身的内容而又不过于强调语言的细节，有些教材采用了伪语言，如类 C 等。但伪语言算法毕竟还要转换成程序语言算法，其中除了语法上的不严格外，伪语句和实际语句上的差别也经常引起程序问题，如调用程序和被调用程序间的信息是双向传递还是单向传递等。

本书对数据结构的描述采用了 C/C++语言，即主体部分为 C 语言，但对输入输出、宏、注释、动态内存分配和释放等采用了比较简洁的 C++扩展形式，见表 1.1 所示。所以上机时需采用 C++编译器（如 Turbo C++ 3.0 等）。C++的其他扩展如引用、类等本书暂未采用。

表 1.1 C/C++语句对比

C 语句	C++语句

#include <stdio.h>
…
scanf("%d%c",&i,&ch);
printf("%d%c\n",i,ch);
	#include <iostream.h>
	…
	cin>>i>>ch;
	cout<<i<<ch<<endl;

#define maxsize 100	const int maxsize=100;
for(i=0;i<n;i++) s=s+a[i];/*数组元素求和*/	for(i=0;i<n;i++) s=s+a[i];//数组元素求和

int *p,*q;
p=malloc(10*sizeof(int));
q=malloc(sizeof(int));
…
free(p);
free(q);
	int *p,*q;
	p=new int[10]; //动态分配 10 个整数空间(数组)
	q=new int; //动态分配 1 个整数空间
	…
	delete []p; //释放数组空间
	delete q;

实践表明，掌握了数据结构的基本知识和方法后，可以非常方便地应用到面向对象程序设计中。

1.2.7 数据结构

前面介绍了数据结构及其相关的几个概念，但一直没有说究竟什么是数据结构。事实上，对数据结构这一概念，目前也没有一致公认的定义。比较流行的观点有两种。一种观点认为，一个数据结构是由一个逻辑结构 S、一个定义在 S 上的基本运算集 Δ 和 S 的一个存储实现 D 所构成的整体(S,Δ,D)；另一种观点认为，一个数据结构是由一个逻辑结构 S 和定义在 S 上的一个基本运算集 Δ 构成的整体(S,Δ)。本书采用前一种观点，将数据的逻辑结构、数据的存储结构及数据的运算这三方面看成一个有机的整体，这样，数据结构的定义为：

数据结构（Data Structure）是指相互间存在着一种或多种关系的数据元素的集合，它们按照某种逻辑关系组织起来，并用计算机语言，按一定的存储方式存储在计算机的存储器中，同时在这些数据上定义了一个运算的集合。简单地说，一个数据结构就是一类数据

此为试读，需要完整PDF请访问：www.ertongbook.com