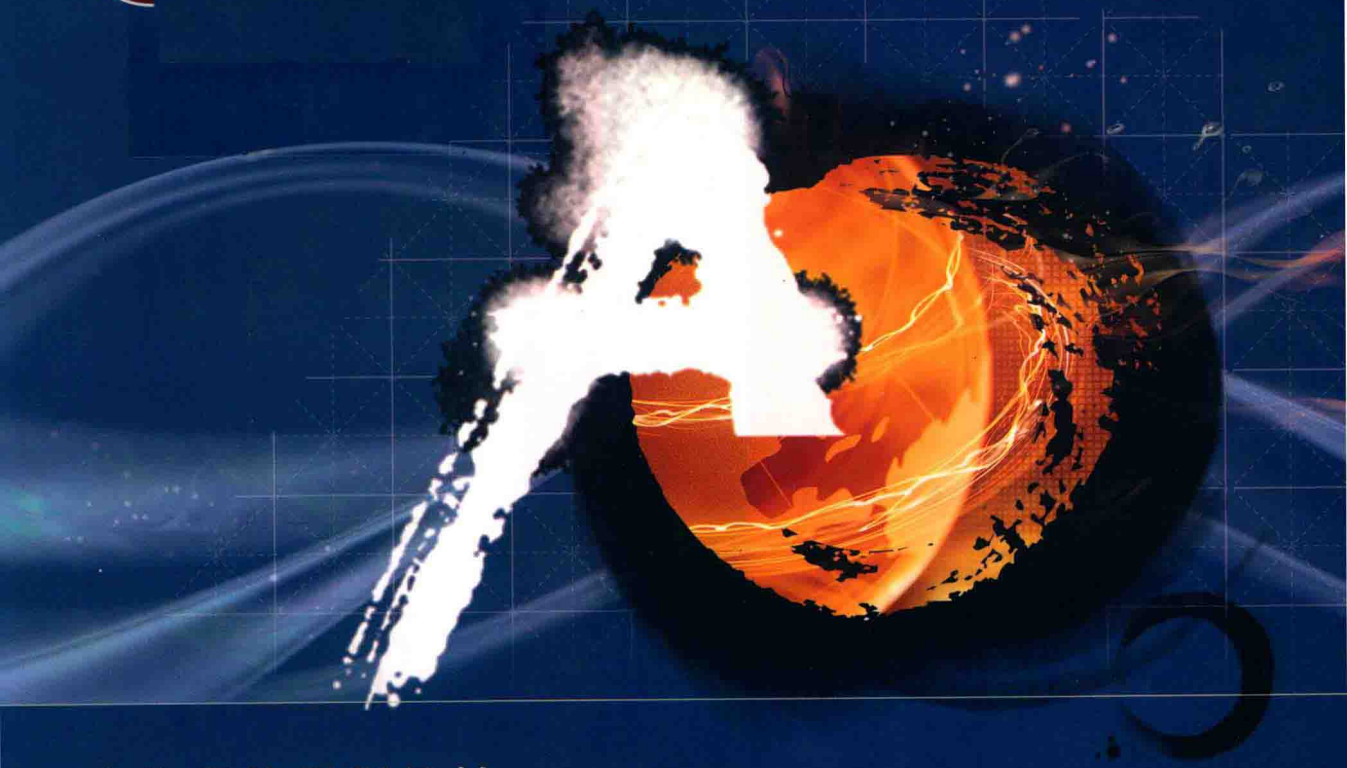
精品教材
U JINGPIN JIAOCAI



高等学校规划教材

C++与数据结构

（第4版）

◎高飞 主编 ◎白霞 胡进 吴浩 聂青 副主编

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

C++与数据结构
（第4版）

高 飞 主编
白 霞 胡 进 吴 浩 聂 青 副主编

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书是国家级（网络教育）精品课程的教学成果，也是北京市高等教育精品教材，根据教育部高等学校大学计算机课程教学指导委员会《大学计算机基础课程教学基本要求》中有关理工类专业的计算机基础课程教学要求组织编写而成，内容由浅入深，案例丰富，通俗易懂，实用性强。

本书在介绍了C++语言的程序设计方法的基础上，采用面向对象的思想和抽象数据类型的概念，用C++语言有效地组织和描述了线性表、堆栈、队列、树和图等各种典型的数据结构和相关类的实现，并介绍了每一种数据结构的不同存储方法、典型操作及其应用。

全书共11章，包括数据结构的基本概念，数组与指针，函数，C++编程基础，继承和多态，模板和STL，线性表，堆栈与队列，树与二叉树，图，查找与散列结构，排序等。本书各章配有习题和实验训练题，方便实践教学，并为任课教师提供了电子课件和示例源代码。

本书可作为高等院校电子信息类以及其他相关专业本科生教材和教学参考书，也可供从事程序设计的工程人员参考使用。

图书在版编目（CIP）数据

C++与数据结构 / 高飞主编. —4版. —北京：电子工业出版社，2018.2
ISBN 978-7-121-31579-4

Ⅰ. ①C… Ⅱ. ①高… Ⅲ. ①C语言－程序设计－高等学校－教材②数据结构－高等学校－教材
Ⅳ. ①TP312.8②TP311.12

中国版本图书馆CIP数据核字（2017）第116800号

策划编辑：章海涛
责任编辑：裴　杰
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编　100036
开　本：787×1092　1/16　　印张：22.75　　字数：582.4千字
版　次：2006年9月第1版
　　2018年2月第4版
印　次：2018年2月第1次印刷
定　价：52.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：192910558（QQ群）。

前　言

本书是**北京市高等教育精品教材**，是**北京市精品课程和国家级精品课程**“数据结构与算法设计”的**配套教材**。编写者是**国家级优秀教学团队**和**北京市优秀教学团队“计算机公共课教学团队”**的主要成员。本书的编写以教育部高等学校大学计算机课程教学指导委员会《大学计算机基础课程教学基本要求》中有关理工类专业的计算机基础课程教学要求为指导思想，借鉴了教育部大学计算机课程改革项目关于培养计算思维的相关成果，结合具体教学改革实践，总结了国家级精品课程和北京市精品课程“数据结构与算法设计”的建设经验，坚持以培养学生解决实践问题的能力为特色，以适应信息时代的人才培养模式。

1. 本书的写作特点

本书是《C++与数据结构（第3版）》的修订版，其写作在延续第3版的特点的基础上，根据实际教学的要求，做了相应的改进。数据结构是计算机算法的设计基础，在计算机科学中占有非常重要的地位。深入研究数据结构对构造完美算法结构和设计具有重要的作用。程序设计语言是实现算法的载体，语言只有满足算法实现的需求，才能被认识和掌握，数据结构只有通过程序语言才能在应用中发挥作用。因此，本书力求以算法为中介，以实现读者学习程序设计语言和学习数据结构的共同进步。

本书在介绍了C++程序设计方法的基础上，采用C++程序设计语言描述算法。C++是一种既支持面向过程程序设计，又支持面向对象程序设计的混合型语言，它独特的面向对象特征，可以为面向对象技术提供全面支持，是描述算法的一种较为理想的语言。采用面向对象程序设计语言描述施加于数据结构之上的算法，不仅有利于与面向对象技术相结合，也为上机实践提高高级语言程序设计水平提供了方便。

本书共包括11章。第1章是对数据结构和面向程序设计方法的概述。第2章～第5章是C++语言程序设计基础，主要内容是C++编程基础，精炼地介绍了数组与指针、函数、C++类及其对象的封装性、引用、友元和重载、继承与派生、多态性与虚函数、模板以及STL的相关内容。第6章～第11章重点介绍典型的数据结构，主要内容包括线性表、堆栈与队列、树与二叉树、图、查找与散列结构、排序。全书每章都配有习题以及相应的程序例题和实验训练题。

本次改版修订，针对读者特点和计算机教学的要求，加强了对编程基础的介绍。在介绍数据结构的同时，在合适的时机引入相关的编程技术的基本原理、原则、实用技巧，以期培养读者良好的编程风格、编程思路，为今后继续深入学习和应用高级编程技巧打下基础。

2. 本书的编排特点

1）本书每章安排有**习题和实验训练题**，可方便实践教学。

2）书中重要内容采用黑体标注。

3）本书**强调可读性**。书中程序全部采用统一的设计风格。例如，类名和变量名的定义做到“望名知义”；采用缩进格式组织程序；对程序中的语句尽可能多地使用注释。

4）本书包含大量的程序示例，给出了运行结果。凡是程序开头带有程序名编号的程序，都是可以直接在计算机上编译运行的程序。

3. 教学支持

本书的电子教案可以在讲课时用多媒体投影演示。教师不仅可以使用本教案，还可以方便地修改和重新组织其中的内容以适应自己的教学需要。使用本教案可以减少教师备课时编写教案的工作量，从而提高教学效果和效率。

本书为教师免费提供电子课件和程序示例源代码。需要的教师可以直接登录华信教育资源网 http://www.hxedu.com.cn 注册后免费下载。

本书由高飞设计总体架构。胡进编写第 1 章和第 6 章，并负责各章节中编程风格、代码改进等；第 2～5 章由白霞编写；第 7 章和第 8 章由高飞编写；第 9 章由聂青编写；第 10 章和第 11 章由吴浩编写。全书由高飞、吴浩统稿、定稿。

感谢北京理工大学“计算机公共课教学团队”中从事教学工作多年的同仁对本书的建议和帮助。电子工业出版社对本书的编写提出了宝贵的意见，在此表示衷心的感谢。

由于计算机算法和程序设计技术发展迅速，编者水平有限，书中的疏漏与不足在所难免，敬请广大读者和同仁不吝赐教，拔冗指正。

感谢大家选用本书，欢迎大家对本书提出意见和建议，编者 E-mail：gaofei@bit.edu.cn。

编　者

于北京理工大学

目　录

第 1 章　数据结构的基本概念

本章学习目标

通过对本章内容的学习，学生应该能够做到：

1）了解：数据结构在计算机数据处理中的作用；基于面向对象描述数据结构算法的优势，以及“对象=数据结构+算法”的概念。

2）理解：数据结构研究的数据之间的逻辑关系、数据在计算机内部的存储结构，以及在数据的各种结构上实施有效的操作或处理（算法）等概念和相互关系。

3）掌握：数据结构的相关基本概念与术语；面向对象的基本概念和基本思想。

计算机已经深入到人类社会的各个领域，计算机的应用已不再局限于科学计算，而是更多地用于控制、管理及数据处理等非数值计算的处理工作。与此相应，计算机加工处理的对象由纯粹的数值发展到字符、表格和图像等各种具有一定结构的数据，这就给程序设计带来了一些新的问题。为了编写出一个好的程序，必须分析待处理对象的特性以及它们之间存在的关系，这就是“数据结构”这门学科形成和发展的背景。分析数据对象之间的逻辑关系，并用计算机存储结构体现出这些逻辑结构并操作这些数据，就是数据结构这门课程要解决的问题。

1.1　数据结构的概念和术语

在讨论数据结构之前，让我们先来介绍几个与数据结构密切相关的概念和术语。

数据（Data）：数据是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机处理的符号的总称。它是信息的载体，是计算机程序加工的原料。对计算机科学而言，数据的含义极为广泛。一般来说，数据主要有两大类，一类是数值数据，包括整数、实数、复数等，主要用于工程、科学计算和商业事务处理；另一类是非数值数据，主要包括字符、字符串、图像、声音等，它们可以通过编码而转变为可被计算机处理的数据。

数据元素（Data Element）：数据元素是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一个**数据元素**可由若干个**数据项**组成，例如，一个学生的基本信息为一个数据元素，可以包括学号、姓名、性别、年龄、成绩、家庭地址等多个数据项。数据项是数据处理中不可分割的最小单位。

数据对象（Data Object）：数据对象是性质相同的数据元素的集合，是数据的一个子集。例如，英文字母数据对象可以是集合 L={‘A’、‘B’、‘C’ …… ‘Z’}，整数数据对象可以是集合 N={–32767，–32766，…，–1，0，1，2，…，32768}。

数据结构（Data Structure）：数据结构是相互之间存在一种或多种特定关系的数据元素的集合。任何问题中，数据元素都不是孤立存在的，它们之间存在某种关系，这种数据元素相互之间的关系称为**结构**。

根据数据元素之间关系的不同特性，通常有如图 1.1.1 所示的四类基本结构。

（1）集合结构

这种结构中的数据元素是无序且没有重复的元素，它们之间除了“同属于一个集合”的

关系外，无其他关系。

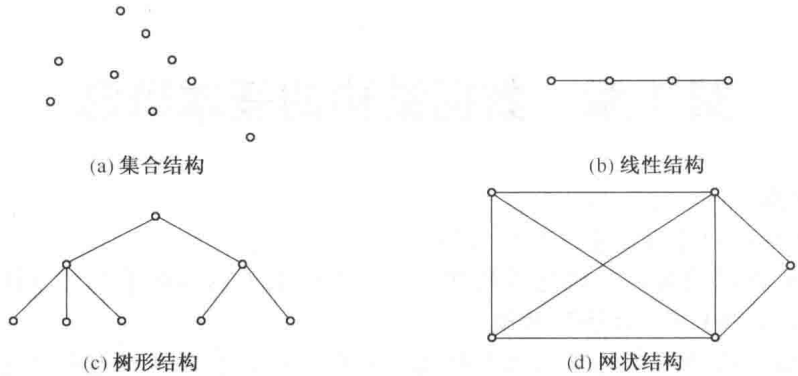


图 1.1.1　四类基本结构

（2）线性结构

这种结构中的数据元素之间存在一个对一个的关系，所有的数据成员按某种次序排列在一个序列中，除第一个元素外，每个元素都有一个且仅有一个直接前驱，第一个数据元素没有直接前驱；除最后一个元素外，每个元素都有一个且仅有一个直接后继，最后一个数据元素没有直接后继。

（3）树形结构

这种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

这种结构中的数据元素之间存在多个对多个的关系。

数据结构是一个二元组：

$$\text{Data\_Structure} = (D, S)$$

其中，D 是数据元素的有限集，S 是 D 上关系的有限集。

上述结构定义中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在计算机中实现对它的操作，因此，我们还需研究如何在计算机中表示它。

数据结构在计算机中的表示称为数据的**物理结构**，又称为**存储结构**，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的一位，叫做位。在计算机中，我们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，通常称这个位串为**元素或结点**。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**。因此，元素或结点可看做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**，对应两种不同的存储结构，即**顺序存储结构和链式存储结构**。顺序映像的特点如下：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。非顺序映像的特点如下：借助指示元素存储地址的指针表示数据元素之间的逻辑关系。数据的逻辑结构和物理结构是密切相关的两个方面。一个算法的设计取决于问题的逻辑结构，而算法的实现依赖于采用的存储结构。

通常我们讨论数据结构时，不但要讨论各种在解决问题时可能遇到的典型的逻辑结构，还要讨论这些逻辑结构的存储映像（存储实现），此外，还要讨论这种数据结构的相关操作及其实现。因此，数据结构要研究的主要内容可以简要地归纳为以下三个方面。

1）研究数据之间固有的客观联系（**逻辑结构**）。

2）研究数据在计算机内部的存储方法（**存储结构**）。

3）研究如何在数据的各种结构上实施有效的操作或处理（**算法**）。

1.2 抽象数据类型

数据结构要研究逻辑结构和存储结构，那么，如何描述存储结构呢？存储结构涉及数据元素及其关系在存储器中的物理表示，由于本书是在高级语言的层次上讨论数据结构的，因此不能直接用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，可用一维数组类型来描述顺序存储结构，用指针来描述链式存储结构。下面来回顾一下什么是数据类型。

1.2.1 数据类型

数据类型是一组性质相同的值的集合以及定义于这个值集合上的一组操作的总称。

计算机处理的数据是以某种特定的形式存在的，如整数、浮点数、字符等形式。C++可以使用的数据类型有以下几种：

C++没有统一规定各类数据的精度、数值范围和在内存中所占的字节数，计算机所能表示的实际数据范围根据编译器和计算机系统结构不同而不同。表 1.2.1 是 Visual C++数值型和字符型数据在内存中所占的字节数和数值范围。

表 1.2.1 数值型和字符型数据的字节数和取值范围

类型	类型标识符	字节	数 值 范 围
整型	[signed] int	4	−2147483648～+2147483647
无符号整型	unsigned [int]	4	0～4294967295
短整型	short [int]	2	−32768～+32767
无符号短整型	unsigned short [int]	2	0～65535
长整型	long [int]	4	−2147483648～+2147483647
无符号长整型	unsigned long [int]	4	0～4294967295
字符型	[signed] char	1	−128～+127
无符号字符型	unsigned char	1	0～255
单精度型	float	4	3.4×10^{-38}～$3.4\times10^{+38}$
双精度型	double	8	1.7×10^{-308}～$1.7\times10^{+308}$
长双精度型	long double	8	1.7×10^{-308}～$1.7\times10^{+308}$

1）整型数据有长整型（long int）、一般整型（int）和短整型（short int）之分。C++没有规定每一种数据所占的字节数，只规定 int 型数据所占的字节数不大于 long 型数据，不小于 short 型数据。一般而言，在 16 位机的 C++系统中，short 型数据和 int 型数据占 2 字节，long 型数据占 4 字节；在 32 位机的 C++系统中，short 型数据占 2 字节，int 型数据和 long 型数据占 4 字节。

2）整型数据以二进制形式存储，如十进制数 65 的二进制为 01000001，在内存中的存储形式如图 1.2.1 所示。

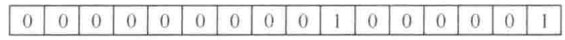


图 1.2.1　整型数据的存储形式

3）整型数据和字符型数据都有带符号和无符号两种形式，分别由修饰符 signed 和 unsigned 表示。如果指定为 signed，则数值以补码形式存放，存储单元最高位表示数值的符号。如果指定为 unsigned，则数值没有符号，全部二进制都用来表示数值本身。占 2 个字节带符号短整型和无符号短整型的存储情况如图 1.2.2 所示。

图 1.2.2　有无符号短整型数据的存储情况

4）浮点数有单精度（float）、双精度（double）和长双精度（long double）之分。在 Visual C++ 6.0 中，float 有 6 位有效数字，double 有 15 位有效数字；float 占 4 字节，double 和 long double 占 8 字节。

数据类型不但规定了使用该类型时的取值范围，还规定了该类型可以使用的一组操作。例如，与整型有关的操作有+、–、*、\、%等。C++不但定义了一些基本的数据类型，还提供了复合的数据类型，如数组、结构体、共用体、类等，程序员可以利用这些复合的数据类型，自行定义一些实际所需要的数据类型，例如，程序员可定义自己的线性表类、栈类等数据类型。

1.2.2　数据抽象与抽象数据类型

在面向对象的程序设计中，常常提到“抽象”一词，那么，什么是抽象呢？抽象的本质就是抽取反映问题本质的东西，忽略非本质的细节。

抽象数据类型通常是指由用户定义，用来表示应用问题的**数据模型**。抽象数据类型由基本的数据类型组成，并包括一组相关的操作，抽象数据类型类似于 C++中的类。对于一个数据成员完全相同的数据类型，如果给它定义不同的功能，则可形成不同的抽象数据类型。

抽象数据类型的特点是使用与实现分离，实行封装和信息隐蔽。在抽象数据类型设计时，把类型的声明与其实现分离开来。首先根据问题的要求，定义该抽象数据类型需要包含哪些信息，并根据功能确定公共界面的服务，使用者可以使用公共界面的服务对该抽象数据类型进行操作。此外，抽象数据类型的具体实现作为私有部分封装在其实现模块内，使用者不能

看到，也不能直接操作该类型所存储的数据，只能通过界面中的服务来访问这些数据。

从实现者的角度来看，把抽象数据类型的具体实现封装起来，有利于编码、测试，也有利于将来的修改。因为这样做可以使得错误局部化，一旦出现错误，其传播范围不至于影响其他模块，如果为了提高效率希望改进数据结构，可能需要改变抽象数据类型的具体实现，但只要界面中的服务的使用方式不变，其他所有使用该抽象数据类型的程序都可以不变，从而大大提高了系统的稳定性。

从使用者的角度来看，只要了解该抽象数据类型的规格说明，就可以利用其公共界面中的服务来使用这个类型，而不必关心其物理实现，这样使用者可以在开发过程中抓住重点，集中精力考虑如何解决应用问题，使问题得到简化。例如，我们在求解一个最优化问题时常常要使用一个栈，那么，我们应当首先考虑此栈应存放什么信息、应如何组织，至于栈怎样实现、可能会出现哪些例外情况、这些例外情况如何处理等，可以忽略，直接调用堆栈类提供的相关服务即可。

1.3 算法和算法分析

1.3.1 算法

数据结构除了要研究数据的逻辑结构和存储结构外，还要研究如何在数据的各种结构上实施有效的操作或处理，这就涉及算法。算法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。一个算法应具有下列五个重要特性。

1）**有穷性**：对任何合法的输入值，一个算法必须在执行有穷步之后结束，且每一步都应该在有穷时间内完成。

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，在任何条件下，算法只有唯一的一条执行路径，对于相同的输入只能得出相同的输出。

3）**可行性**：一个算法是可行的，即算法中描述的操作都可以通过已经实现的基本运算执行有限次来实现。

4）**输入**：一个算法有 0 个或多个输入，这些输入取自于某个特定的数据对象的集合，它可以使用输入语句从外部提供，也可以在算法内通过赋初值给定。

5）**输出**：一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

1.3.2 算法设计的要求

要高质量、高效率地完成好的算法或好的代码，需要编程人员养成好的编程习惯、编程风格。通常一个好的算法应考虑达到以下目标。

1. 正确性

算法应当满足具体问题的需求。通常一个大型问题的需求，要以特定的规格说明方式给出，而一个实习问题或练习题，往往就不那么严格。目前大多采用自然语言描述需求，它至少应当包括对于输入、输出和加工处理等明确的无歧义性的描述。设计或选择的算法应当能满足这种需求。

正确性大体可分为以下 4 个层次。

1）程序不含语法错误。

2）程序对于几组输入的数据能够得出满足规格说明要求的结果。

3）程序对于精心选择的、典型的、苛刻的几组输入数据能够得出满足规格说明要求的结果。

4）程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

要达到第四层含义下的正确是极为困难的，所有不同输入数据的数据量大得惊人，逐一验证的方法是不现实的。对于大型软件需要进行专业测试，而在一般情况下，通常以第3层意义的正确性作为衡量一个程序是否合格的标准。

为保证程序的正确性，需要掌握一定的测试、调试技巧。利用这些技巧，可以事半功倍地发现、排除编码错误，因而，当使用一个编程工具或开发环境时，要注重对其提供的调试功能的了解和使用。一个虽然简单但是非常有用的基本调试方法是对代码设置断点，使得代码运行到断点处停止，这样，程序的整个运行过程将不再是一个不可控的黑匣子。当程序暂停于断点处时，编程者可以查看各种变量或内存状况从而帮助自己分析代码问题，保证程序的正确性或其他质量。

2. 可读性

算法主要是为了人的阅读与交流，其次才是机器执行。可读性好有助于人对算法的理解，晦涩难懂的程序易于隐藏较多错误，往往难以调试和修改。现代软件注重代码的复用，以期更加高效地对软件进行扩展。良好的可读性，不但是代码团队协作开发的需要，也是代码复用的需要。可读性涉及很多方面，如算法本身的逻辑是否清晰易懂，程序结构是否明晰合理，等等。但是作为初学者，可以从一些基本的细节开始，注重培养保持代码可读性的习惯。例如，对函数、重要变量的命名，尽量含义清晰，遵守变量命名的一些习惯（例如，指针前面加“p”），合理地加上代码注释，等等。一些语法规则的运用，既有利于代码的安全性，也能提高代码的可读性。例如，在函数传递的参数前，如果加上了 const 限定，既可以防止函数内部错误地修改该变量，又可告诉代码阅读者此变量在函数体内不需要改变，因而提高了代码的可读性。后面将结合代码示例，适当介绍提高可读性的相关知识。

3. 健壮性

当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫名其妙的结果，处理错误的方法可以是返回一个表示错误或错误性质的值。除了对非法输入的容错之外，健壮性对其他异常情况，如内存异常、磁盘文件异常、网络异常等也要有容错性。另外，健壮性还包括对程序长期稳定运行的要求。有些程序启动后需要长时间运行，因此如果没有很好地管理内存、资源，将会导致程序消耗的内存或资源越来越多,从而引起程序运行迟缓、系统崩溃等问题。还有些程序没有很好地管理产生的临时文件，也会导致时间长了以后运行迟缓、占用存储空间过多等问题。所以，良好的编程的风格和习惯，对提高代码的健壮性很有帮助。

4. 效率

效率指的是算法执行时计算机资源的消耗，它包括运行时间代价和存储空间代价。对于同一个问题，如果有多个算法可以解决，则执行时间短、存储量需求小的算法效率高。效率与问题的规模和性质有关，求 10 个人的平均工资与求 1000 个人的平均工资所花的执行时间或运行空间显然有一定的差别。有时候，程序的效率与可读性、健壮性等是有竞争的，这时需要结合使用背景、程序生命周期等因素进行综合考虑，取得平衡。接下来，我们将重点讨论算法的效率。

1.3.3 算法效率的度量

算法的效率包括算法运行时间代价和存储空间代价，它们分别由**时间复杂度和空间复杂度**来度量。

1. 算法的时间复杂度

算法执行时间需依据该算法编制的程序在计算机上运行时所消耗的时间来度量。而度量

一个程序的执行时间通常有两种方法，即事后统计的方法和事前分析估算的方法。

（1）事后统计的方法

很多计算机内部都有计时功能，有的甚至可精确到毫秒级，不同算法的程序可通过一组或若干组统计数据来分辨优劣。但这种方法有两个缺陷，一是必须先运行依据算法编制的程序；二是统计数据依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。因此，人们常常采用另一种事前分析估算的方法。

（2）事前分析估算的方法

一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素。

① 问题的规模。

② 算法选用的策略。

③ 书写程序的语言，对于同一个算法，实现语言的级别越高，执行效率就越低。

④ 编译程序所产生的机器代码的质量。

⑤ 机器执行指令的速度。

显然，同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行时，效率均不相同。这表明使用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为，一个特定算法的运行时间，只依赖于问题的规模，或者说，它是问题规模的函数。

一个算法是由控制结构（顺序结构、分支结构和循环结构）和原操作（指固有数据类型的操作）构成的。算法的时间取决于两者的综合效果。为了便于比较同一问题的不同算法，通常的做法是，从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该基本操作的重复执行的次数作为算法的时间度量。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数 $f(n)$，算法的时间量度记为

$$T(n) = O(f(n)) \tag{1.3.1}$$

式（1.3.1）表示随问题规模 n 的增大，算法执行时间的增长率和 $f(n)$ 的增长率相同，$T(n)$ 称为算法的渐近时间复杂度，简称**时间复杂度**。

在大多数情况下原操作是最深层循环内的语句中的原操作，它的执行次数和包含它的语句的频度相同（语句的频度指的是该语句重复执行的次数）。

例如：

1）
```
++ x;
```

2）
```
for ( i = 0; i < n; i ++) x ++;
```

3）
```
for ( i = 0; i < n; i ++)
        for ( j=0; j < n; j ++)
            x ++;
```

以上三例中，含基本操作“x ++”的语句的频度分别为 1、n 和 n^2，这三个程序段的时间复杂度相应为 $O(1)$，$O(n)$ 和 $O(n^2)$，分别称为常量阶、线性阶和平方阶。算法还可能呈现的时间复杂度有对数阶 $O(\log n)$、指数阶 $O(2^n)$ 等。不同数量级时间复杂度的性状不同，我们应该尽可能选用多项式阶 $O(n^k)$ 的算法，而不希望用指数阶的算法。算法复杂度示意图如图 1.3.1 所示。

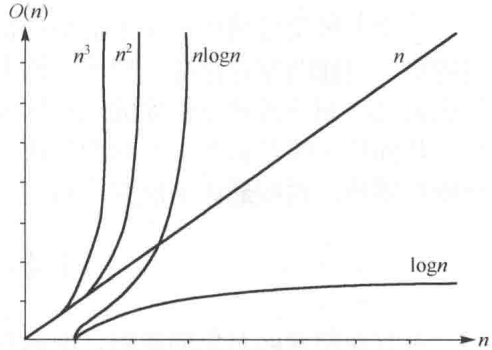


图 1.3.1　算法复杂度示意图

一般情况下，对一个问题（或一类算法）只需选择一种基本操作来讨论算法的时间复杂度即可，有时也需要同时考虑几种基本操作，甚至可以对不同的操作赋以不同的权值，以反映执行不同的操作所需要的相对时间，这种做法便于综合比较解决同一问题的两种完全不同的算法。

有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同，如下面起泡排序法的算法：

```
void bubblesort(int *a, int n)
{   for( i = n-1, flag =1; i >=1&& flag; --i )
    {   flag = 0;
        for( j = 0; j < i; ++ j )
            if( a[ j ] > a[ j+1] )
            {   t = a[ j ], a[ j ] = a [ j+1], a[ j+1] = t;
                flag =1;
            }
    }
}
```

此例中，“交换序列中相邻的两个整数”是基本操作。当数组 a 中初始系列为自小到大有序时，基本操作的执行次数为 0。当初始系列为自大到小有序时，基本操作的执行次数为 $n(n-1)/2$。对这类算法的分析，一种解决方法是计算它的平均值，即考虑它对所有可能的输入数据集的期望值，此时相应的时间复杂度为算法的平均时间复杂度。然而，在很多情况下，各种输入数据集出现的概率难以确定，算法的平均时间复杂度也难以确定。因此，另一种更可行也更常用的办法是讨论算法在最坏情况下的时间复杂度，例如，上述起泡排序的最坏情况为数组 a 中初始序列为自大到小有序，则起泡排序算法在最坏情况下的时间复杂度为 $T(n)=O(n^2)$。在本书以后各章中讨论的时间复杂度，除特别指明外，均指**最坏情况**下的时间复杂度。

实际中我们可以把事前估算和事后统计两种办法结合起来使用。以两个矩阵相乘为例，若上机运行两个 10×10 的矩阵相乘，执行时间为 12ms，则由算法的时间复杂度 $T(n)=O(n^3)$ 可估算两个 20×20 的矩阵相乘所需时间大致为

$$(20/10)^3\times 12=96\text{ms}$$

2. 算法的空间复杂度

类似于算法的时间复杂度，以**空间复杂度**作为算法所需存储空间的量度，记为

$$S(n)=O(f(n)) \tag{1.3.2}$$

式（1.3.1）中，n 为问题的规模（或大小）。

一个上机执行的程序除了需要存储空间来存储本身所用指令、常数、变量和输入数据外，也需要一些辅助空间存储一些所需的中间信息。若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的额外空间，否则应同时考虑输入本身需要的空间。若额外空间是常数，则称此算法为原地工作。如果所占空间量依赖于特定的输入，则除特别指明外，均按**最坏情况**来分析。

1.4 面向对象概述

本节介绍面向对象的思想、面向对象的基本概念以及面向对象程序设计的基本特征。