21世纪高等学校规划教材
Textbook Series of 21st Century

数据结构（C语言版）

王翠茹 袁和金 刘 军 编 著
马玉梅 牛为华

中国电力出版社
http://jc.cepp.com.cn

21世纪高等学校规划教材
Textbook Series of 21st Century

数据结构（C语言版）

编　著　王翠茹　袁和金　刘　军
　　马玉梅　牛为华
主　审　王熙照

中国电力出版社
http://jc.cepp.com.cn

内 容 提 要

本书为21世纪高等学校规划教材。

本书从抽象数据类型的观点出发，系统全面地介绍了数据结构课程中的基本理论、方法及技巧。全书共分绪论、顺序表、链表、数组和广义表、串、树、图、查找表、内排序、文件、外排序以及数据结构的应用示例和一个附录，在每章的末尾配备了足够的习题，附录对实习步骤和内容作了较详细的介绍。

本书可作为普通高等院校计算机相关专业的教材，也可供计算机工程与科技工作人员参考。

图书在版编目（CIP）数据

数据结构：C语言版/王翠茹等编著. —北京：中国电力出版社，2006.9

21世纪高等学校规划教材

ISBN 7-5083-4584-3

Ⅰ.数… Ⅱ.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.TP311.12②TP312

中国版本图书馆CIP数据核字（2006）第084888号

中国电力出版社出版、发行

（北京三里河路6号 100044 http://jc.cepp.com.cn）

北京同江印刷厂印刷

各地新华书店经售

*

2006年8月第一版 2006年8月北京第一次印刷

787毫米×1092毫米 16开本 18.5印张 453千字

印数0001—3000册 定价**28.00**元

前　言

为了有效地使用计算机，设计出高效、准确的程序，首先就需要对数据的性质和数据元素间的关系进行深入地研究。换言之，必须对数据内部的结构关系，以及在计算机内如何表示和操作这种结构的技术有透彻的了解。作者根据多年来讲授《程序设计》、《数据结构》、《算法设计与分析》等课程的教学体会，于1997年编写出版了Pascal版的《数据结构》，并获得了河北省科技进步三等奖，几年来经过征询专家和学生的建议，进行了修改和补充，编写了本书，在内容上力求做到图文并茂、深入浅出，对每个算法都阐述其设计思想和实现方法，并用具体例子加以说明。凡学过一门高级程序设计语言的人，都能看懂书中的内容。具有离散数学和概率论知识的读者，学习本书就更加容易了。

全书共分十二章，第一章通过分析一个实际问题的求解过程，引入抽象数据类型、数据结构和算法等概念，对书中所用程序设计语言和算法分析作了粗略的说明；第二章至第七章分别讨论了顺序表、链表、数组和广义表、串、树和图等常见的几种数据结构的特点、存储方式和基本运算，对每一种结构都举了一些例子来说明这些技术的各种应用；第八章至第十一章分别讨论了常用的查找、排序方法以及文件的物理结构，并对一些简单的算法作了定量的分析；第十二章主要列举了数据结构的一些实际应用示例，使读者对自己所学知识达到进一步深化，起到学以致用的效果。附录中介绍了实习步骤（包括怎样写实习报告）和实习内容。

本书讲授60—90学时，内容可视学时的多少进行取舍，在学时较少的情况下，可以舍去诸如判定树、文件、外排序等内容。第一章至第五章由华北电力大学计算机科学与技术学院袁和金老师编写；第六章至第十二章由华北电力大学计算机科学与技术学院王翠茹教授编写；算法部分由华北电力大学计算机科学与技术学院刘军老师、马玉梅老师和牛为华老师调试。全书由王翠茹教授统稿。河北大学数学与计算机学院王熙照教授认真审阅了全部书稿，并提出了许多宝贵的修改意见，编者表示深切的谢意。

限于作者水平和编写时间，书中必定存在错误和缺点，恳切希望读者批评指正。

作　者

2006年3月

目　录

第一章　绪　　论

程序设计的实质是数据表示和数据处理，而这种表示和处理应通过一个渐进的过程逐步完成。数据结构课程主要讨论这个过程中的一些最基本的问题。本章将概括地介绍有关的基本概念、基本思想以及本课程所用的算法语言，最后给出了度量算法代价的大“O”表示法。

§1.1　引　　言

计算机科学是一门研究数据表示和数据处理的科学，数据则是计算机化的信息，它是计算机可以直接处理的最基本最重要的对象，无论是进行科学计算或数据处理、过程控制以及对文件的存储和检索乃至数据库技术等计算机应用领域中，都是对数据进行加工处理的过程。因此要设计出一个结构好效率高的程序，必须研究数据的特性、数据间的相互关系及其对应的存储表示，并利用这些特性和关系设计出相应的算法和程序。这一过程的归纳和抽象就是数据结构这门学科产生和发展的由来。

“数据结构”作为一门独立的课程在国外是从1968年才开始的，但在此之前其有关内容已散见于“编译原理”及“操作系统”之中了。20世纪60年代中期，美国的一些大学开始设立有关课程，但当时的课程名称并不叫“数据结构”而被称作“表处理语言”，它的主要内容是研究当时已经出现的几种表处理语言。如J. Weizenbaum在20世纪50年代初设计的SLIP语言（简单表处理语言）；50年代后期由A. Newell等设计的IPLV语言（信息处理语言）；1959年到1960年J. McCarthy设计的著名的LISP系统（表处理语言）；以及1962年由D. Farber等人设计的SNOBOL语言（串处理语言）。这些语言系统的共同特点是，处理的数据对象的结构形式或者是表结构，或者是树结构。如LISP语言的数据结构就是二叉树形式，SONBOL的数据结构则是表和树的形式。上述几种语言是以数据为中心，为处理非数值问题而设计的，如LISP就是为实现人工智能而设计的表处理语言。而像FORTRAN、ALGOL等一般人较为熟悉的算法语言则主要是为解决数值计算问题而设计的，它们侧重以建立程序为中心，只有当数据成为程序的加工对象时，才考虑到数据。这种观点显然适用于数值计算问题，这类问题属于在简单数据结构上进行复杂的函数变换问题。以数据为中心的观点则把数据结构作为问题的中心（如数据库），而把程序看成是围绕着数据结构缓慢爬行的小虫，它时而询问，时而修改或扩充当前放在内存中的数据。这种观点适用于诸如人事档案管理系统、航空订票系统以及情报检索系统等非数值计算问题的处理，这些大的系统都要求采用复杂的数据结构描述系统的状态，它们的运算（或操作）是为了实现对于数据结构的访问或修改等。

由于数据必须在计算机中进行处理，因此，除去对数据本身的数学特征需要加以注意和研究之外，还必须考虑数据的物理结构，以及这种数据结构的相关操作及其“合理”实现。

§1.2 问题的求解过程

用计算机解决一个具体问题时，一般情况需要下列几个步骤：

(1) 分析阶段：这个阶段的任务是弄清所要解决的问题是什么，并且把它用某种方式清楚地描述出来；

(2) 设计阶段：这个阶段的任务是算法的设计，模块的设计和数据结构的设计，即建立程序系统的结构；

(3) 编码阶段：这个阶段的主要任务是根据系统结构的要求编写出可执行的程序；

(4) 测试阶段：这个阶段的任务是发现和排除在前几个阶段中产生的错误；

(5) 维护阶段：这个阶段的任务是对程序系统进行运行和管理；

数据结构课程并不研究由上述五个阶段构成的程序开发全过程，而是集中讨论以其中的设计阶段为核心，同时涉及编码和分析阶段的一个小范围内若干基本问题。

例如，用回溯法解决组合锁问题。某种车的一种组合锁由 N 个开关组成。每个开关可以“闭合”或“打开”。假设当⌊N/2⌋或更多一些开关处在“闭合”位置时，就可能打开这个锁。假定我们已经忘记了这个组合，而必须把这个锁打开，就得去试一试所有的组合，用回溯法来找到这个组合。

弄清问题是解决问题的前提。我们可用一个 N 位二进制数模拟每一种可能的组合，如果开关 i 闭合，则第 i 位为 1，如果开关 i 打开，则第 i 位为 0。利用一棵二叉树作为 N 个元素的不同状态的模型，这棵树第 K 级上的每一个节点，将对应于 N 位数中前 K 位的不同位置值。这一级上的每个节点，又向下分成两个节点。这两个节点是相应这个 N 位数在第(K+1)位上两个可能的位置值。本例仅考虑 N=4 时的二叉树结构，如图 1.1 所示。

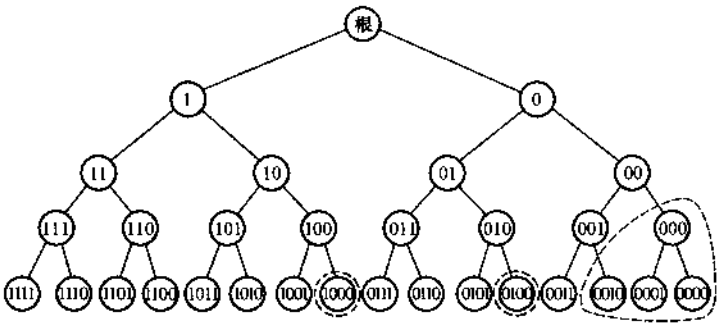

图 1.1 组合锁问题求解树模型

由于至少必须有⌊N/2⌋个开关“闭合”才有可能满足“打开”这个条件，所以对于一些不能提供正确组合的分支便不去生成它。具体来说，如果节点含有两个 0，它的右支就不必去生成。如图 1.1 用细虚线框框起部分，就不必去生成。

下面介绍用二叉树作为组合锁问题的数学模型，生成至少有⌊N/2⌋个开关“闭合”的那些组合的回溯过程。实际上是对这棵树的遍历。首先，从根节点出发，选取这棵树的最左一支，即沿左分支向下直到终端节点，校验这个节点是否符合正确组合的条件。若不符合，则回溯一级。查找是否有没查过的分支，若有，则沿着没查过的分支向下搜索，搜索到终端节点时校验，若不符合打开条件(正确组合条件)便回溯；若所有分支都查过便直接回溯一级，再按同样的方法搜索。对图 1.1 用回溯法按下面的次序访问节点。

根

1
11
111
1111 判断这个组合，若不是，则回溯；否则结束；
111 向下取右分支；
1110 判断这个组合，若不是，则回溯；否则结束
111 左右分支已查完，再回溯；
11 取右分支；
110 取左分支；
1101 判断这个组合，若不是，则回溯；否则结束
110 取右分支；
1100 判断这个组合，若不是，则回溯；否则结束
110 无待查节点，回溯；
11 无待查节点，回溯；
1 取右分支；
10 取左分支；
101 取左分支；
1011 判断这个组合，若不是，则回溯；否则结束
101 取右分支；
1010 判断这个组合，若不是，则回溯；否则结束
101 无待查节点，回溯；
10 取右分支；
100 取左分支；
1001 判断这个组合，若不是，则回溯；否则结束
100 因为 100 的右分支不满足至少有两个 1 这个限制，因此不搜索 100 的右分支，直接回溯到 10；
10 无待查节点，回溯；
1 无待查节点，回溯。
根 取右分支；
⋮

当再次返回到根节点时，或已找到了正确的组合时，算法结束。

从上面例子可见，描述这类非数值计算问题的数学模型不再是数学方程，对上例是用树这种数据结构来描述；如要解决数据库管理方面的问题可用表结构来描述；如要解决旅游花费问题可用图结构来描述。

问题求解的一般过程，可用如图 1.2 所示的框图来概括。

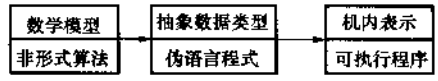

图 1.2 问题求解过程

从图 1.2 看出，要想解决一个实际问题，首先要建立恰当的数学模型来描述要处理的问题，并设计出一个求解该问题的初步的“非形式算法”。然后，用逐步求精的方法对“非形式算法”细化，

用伪语言写出比较形式的算法，对那些在算法中用到的非基本的数据类型建立抽象数据类型，用过程名给这个类型上的每个操作命名，并且用对这些过程的调用来取代算法中的每个操作。最后对每个抽象数据类型选择一种实现的方法，同时编写出这些抽象数据类型上定义的所操作的实现过程。若伪语言程序中还有不可执行的语句，则要加以细化，从而得到的全部是可执行的程序。这一过程的每一步骤都包括数据表示与数据处理两个方面。在上述过程中，数据表示的形式经历了如下变化：机外表示→数学模型→抽象数据类型→机内表示。数据处理方面的变化过程是：处理要求→非形式算法→伪语言程序→可执行程序。

§1.3 基 本 概 念

将本书中常用的名词和术语给出确定的含义。

1. 数据（data）

随着计算机科学的发展和计算机应用的普及，计算机加工处理的对象已从早期的数值、布尔值等扩展到字符串、表格、图像、声音等等。因此，凡是能被计算机存储、加工的对象通称为数据。它是计算机程序加工的“原料”。

2. 数据元素和数据项（data element and item）

数据元素是数据的基本单位，在程序中作为一个整体加以考虑和处理。有时，一个数据元素可由若干个数据项组成，数据项是数据的不可分割的最小单位。

3. 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，整数数据对象是集合 $D=\{0,\pm1,\pm2,\cdots\}$，字母字符数据对象是集合 $C=\{'A','B',\cdots,'Z'\}$。

4. 数据结构（data structure）

数据结构不同于数据类型，也不同于数据对象，它不仅仅涉及到数据类型和数据对象，而且要描述数据对象各元素之间的相互关系。这种数据元素之间的相互关系就称为结构。根据数据元素之间关系的不同，通常有如图 1.3 所示的四类基本的结构形式。

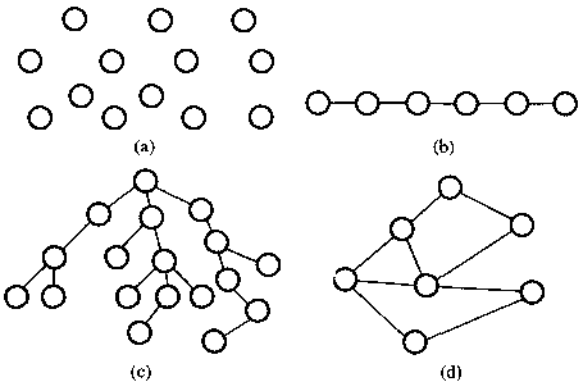

图 1.3 四种基本结构关系图

(a) 集合；(b) 线性结构；(c) 树形结构；(d) 图状结构

(1) 集合：集合中的任何两个节点之间都没有逻辑关系，组织形式松散。

(2) 线性结构：结构中元素之间存在一对一的关系，依次排列形成一条“锁链”。

(3) 树形结构：结构中元素之间存在一对多的关系，具有分支、层次特性。

(4) 图状结构：结构中元素间存在多对多的关系，元素间互相缠绕，任何两个元素都可以邻接。

数据结构可以用二元组的形式来进行数学意义上的形式定义。

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。它是从操作对象中抽象出来的数学模型，仅仅描述了数据元素之间的某种逻辑关系，如何在计

算机中实现对数据元素的操作，还需要考虑在计算机中的存储。因此数据结构应包括如下三方面的内容。

(1) 数据的逻辑结构：数据元素之间的逻辑关系。

(2) 数据的存储结构：数据元素及其关系在计算机存储器中的表示。

(3) 数据的运算：对数据对象施加的操作。

数据的逻辑结构（logic structure of data）是从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看作是从具体问题上抽象出来的数学模型，通常我们把数据元素之间的关联方式（邻接关系）叫做数据元素间的逻辑关系。数据元素之间逻辑关系的整体称为逻辑结构。

数据的存储结构（torage structure of data）是数据结构在计算机中的表示。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数的一位，叫做位(bit)。在计算机中，我们可以用一个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节点（node)。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（data field)。元素间的关系在计算机内的表示方法通常有四种：

(1) 顺序存储方法：该方法是每个存储节点只含有一个数据元素，所有存储节点相继存放在一个连续的存储区里。用存储节点间的位置关系表示数据元素之间的逻辑关系。

(2) 链式存储方法：该方法是每个存储节点不仅含有一个数据元素，还包含一组指针。每个指针指向一个与本节点有逻辑关系的节点。

(3) 索引存储方法：该方法是在存储节点信息的同时，还建立索引表，索引表的每一项称为索引项，索引项的一般形式是：(关键字，地址)。关键字是能惟一标识一个节点的那些数据项，地址表示该关键字所在节点的起始存储位置。

(4) 散列存储方法：该方法是根据节点的关键字，采用某种方法，直接计算出该节点的存储地址。

5. 数据类型（data type)

数据类型是一个值的集合和定义在这个值集上的一组操作的总称。例如C语言中的整数类型，其值集区间为［- Maxint，Maxint］上的整数，定义在其上的一组操作为：加、减、乘、整除和取模等。按“值”的不同特性，数据类型可分为两类：一类是非结构的原子类型。原子类型的值是不可分解的。如C语言中的标准类型（整、实、字符、布尔型)、枚举类型、子界类型和指针类型；另一类是结构类型。结构类型的值是由若干成分按其某种结构组成的，因此其值是可分解的，并且它的成分可以是非结构的，也可以是结构的。例如一个记录类型的元素的值由若干个分量组成，每个分量可以是整数，也可以是数组，也可以是指针类型等等。引入“数据类型”的目的仅从使用数据类型的角度来说，实现了信息的隐蔽，将一切不必了解的细节都封装在类型中。例如，用户在使用“整数”类型时，既不需要了解“整数”在计算机内的表示，也不需要知道其操作是如何实现的。如“两整数求和”程序设计者注重的仅仅是其“数学上求和”的抽象特性，而不是其硬件的“位”操作如何进行。

6. 抽象数据类型（Abstract Data Type)

抽象数据类型和数据类型实质上是一个概念，是指一个数学模型以及定义在该模型上的

一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内的表示和实现无关，即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。抽象数据类型与数据类型相比范畴更广些，它不再局限于前述程序语言中已定义并实现的数据类型（固有的数据类型），还包括用户在设计软件系统时自己定义的数据类型。

假设要设计一个圆（Circle）抽象数据类型，它提供的操作除了按给定半径构造一个圆外，还要包括计算面积（area）和周长（circumference）的操作。下面是 Circle 的一个界面定义，关于这个类型的具体实现（包括数据结构的表示和完成操作的代码）请读者自己完成。

ADT circle is
  data
    非负实数表示圆的半径
  operations
    constructor
      输入的初值：非负实数
      处理：构造一个圆
    area
      输入：无
      输出：圆的面积
      处理：计算圆的面积
    circumference
      输入：无
      输出：圆的周长
      处理：计算圆的周长
end ADT circle

和数据结构的形式定义相对应，抽象数据类型可以用以下三元组表示：

$$ADT = (D, S, P)$$

式中：D 是数据对象，用节点的有限集合表示；S 是 D 上的关系集，用节点间序偶的集合表示；P 是对 D 的基本操作集。其操作的定义格式为：

基本操作名（参数表）
初始条件：<初始条件描述>
操作结果：<操作结果描述>

例：构造三元组表

ADT Triplet
{
对象：$D=\{e_1, e_2, e_3 \mid e_1, e_2, e_3 \in ElemSet\}$
关系：$R1=\{<e_1, e_2>, <e_2, e_3>\}$
操作：InitTriplet(&T, v_1, v_2, v_3)
操作结果：构造三元组 T，元素 e_1, e_2, e_3 分别被赋以参数 v_1, v_2, v_3 的值
操作：Get(T, i, &e)取表中元素
}

至于上述抽象类型如何具体实现，读者可以根据所学过的知识来思考一下，在这里就不赘述了。

7. 基本运算（basic operation）

基本运算是只描述处理功能，不包括处理步骤和方法，是在逻辑结构上的操作。

8. 运算实现（realization of operation）

运算实现的核心是处理步骤的规定，即算法设计。对一个复杂的运算仍需按照逐步求精的方法构造它的实现。

9. 算法（algorithm）

算法是指解决某一特定类型问题的有限运算序列。一个算法应该具有下列特性：

（1）有穷性：一个算法必须总是在执行有穷步之后结束；

（2）确定性：算法的每一步，必须有确切的定义，读者理解时不会产生歧义；

（3）输入：一个算法有一个或多个输入，这些输入取自于特定的对象集合；

（4）输出：一个算法有一个或多个输出，它们是同输入有某种特定关系的量；

（5）可行性：算法应该是能行的，即算法中所有待实现的运算都是能够正确进行的。

§1.4 数据结构课程内容

数据结构与数学、计算机硬件和软件有十分密切的关系。数据结构是介于数学、计算机硬件和计算机软件之间的一门计算机科学专业的核心课程，是高级程序设计语言、编译原理、操作系统、数据库、人工智能等课程的基础。同时，数据结构的技术也广泛应用于信息科学、系统工程、应用数学以及各种工程技术领域。

数据结构课程集中讨论软件开发过程中的设计阶段，同时涉及编码和分析阶段的若干基本问题。此外，为了构造出好的数据结构及其实现，还需考虑数据结构及其实现的评价与选择。因此，数据结构课程的内容包括三个层次的五个“要素”，其内容体系如图 1.4 所示。

层次 \ 方面	数据表示	数据处理
抽象	逻辑结构	基本运算
实现	存储结构	算法
评价	不同结构的比较及算法分析	

图 1.4 数据结构课程内容体系

§1.5 类 C 语言和算法分析

1.5.1 类 C 语言

本书采用的类 C 语言精选了 C 语言的一个核心子集，同时做了若干扩充修改，增强了语言的描述功能。以下对其做简要说明。

（1）预定义常量和类型：

```
//函数结果状态代码
#define TRUE 1
#define FALSE 0
#define INFEASIBLE -1
#define OVERFLOW -2
```

```
//Status 是函数的类型,其值是函数结果状态代码
typedef int Status;
```

(2) 数据结构的表示(存储类型)用类型定义(typedef)描述。数据元素类型约定为ElemType,由用户在使用该数据类型时自行定义。

(3) 基本操作的算法都用以下形式的函数描述:

```
函数类型  函数名(函数参数表)
{  //算法说明
   语句序列
}  //函数名
```

除了函数的参数需要说明类型外,算法中使用的辅助变量可以不做变量说明,必要时对其作用给予注释。一般而言,a、b、c、d、e等用作数据元素名,i、j、k、l、m、n等用作整形变量名,p、q、r等用作指针变量名。但函数返回值为函数结果状态代码时,函数定义为Status类型。

为了便于描述,在函数表中除了值调用方式外,增添了C++语言的引用调用的参数传递方式。在形参表中,以&打头的参数即为引用参数。引用参数能被函数本身更新参数值,可以此作为输出数据的管道。参数表中的某个参数允许预先用表达式的形式赋值,作为默认值使用,以简化参数表。

(1) 内存的动态分配与释放:

使用new和delete动态分配和释放内存空间:

分配空间 指针变量=new 数据类型。

释放空间 delete 指针变量。

(2) 赋值语句有:

```
简单赋值    变量名=表达式;
串联赋值    变量名1=变量名2=…=变量名k=表达式;
成组赋值    (变量名1,…,变量名k)=(表达式1,…,表达式k);
            结构名=结构名;
            结构名=(值1,值2,…,值k);
            变量名[]=表达式;
            变量名[起始下标…终止下标]=变量名[起始下标…终止下标];
交换赋值    变量名←→变量名;
条件赋值    变量名=条件表达式? 表达式T:表达式F。
```

(3) 选择语句有:

```
条件语句1   if(表达式) 语句;
条件语句2   if(表达式) 语句;
            else  语句;
开关语句1   switch(表达式)
            {  case  值1:语句序列1;break;
               case  值2:语句序列2;break;
               …
```

```
            case  值 n:语句序列 n;break;
            default:语句序列 n+1;
        }
开关语句 2  switch
        {   case  条件 1:语句序列 1;break;
            case  条件 2:语句序列 2;break;
            …
            case  条件 n:语句序列 n;break;
            default:语句序列 n+1;
        }
```

(4) 循环语句有:

for 语句　for(赋值表达式序列;条件;修改表达式序列)语句;

while 语句　while(条件)语句;

do-while 语句　do

{　语句序列;

}while(条件)。

(5) 结束语句有:

函数结束语句　return 表达式;

return;

case 结束语句　break;

异常结束语句　exit(异常代码)。

(6) 输入和输出语句有:

输入语句　scanf([格式串],变量 1,…,变量 n)

输出语句　printf([格式串],表达式 1,…,表达式 n)。

(7) 注释:

单行注释　//文字序列。

(8) 基本函数有:

求最大值　max(表达式 1,…,表达式 n);

求最小值　min(表达式 1,…,表达式 n);

求绝对值　abs(表达式);

求进位整数值　ceil(表达式);

求不足整数值　floor(表达式);

判定文件结束　eof(文件变量)或 eof;

判定行结束　eoln(文件变量)或 eoln。

(9) 逻辑运算约定:

与运算 &&　对于 A&&B,当 A 的值为 0 时,不再对 B 求值;

或运算 ||　对于 A||B,当 A 的值为非 0 时,不再对 B 求值。

1.5.2 如何评价一种算法

解决同一问题,常常可以设计出不同的算法,如何比较这些算法的优劣是一个有意义的

问题。如本书第八章对一个排序问题就介绍了多种不同的方法，这些方法的排序速度和占用内存的多少存在着很大的差异。评价算法的目的，既是为了从解决同一问题的不同算法中选出较为适用的一种，同时也有助于考虑对现有的算法如何进行改进或设计出新的算法。

1. 评价算法的一般原则

一个好的算法通常来说主要应具有如下特点：

(1) 正确性：这是指算法在允许的输入数据范围内，能在有穷时间内得出正确的结果。关于算法正确性的证明需要使用有关的数学理论（如集合论、代数学的定理及离散数学等知识），严格地说它属于“程序设计方法学”课程所研究的内容。通常，对于较复杂的问题，可以将算法分解成一些局部的模块来分析，只有每个模块都是正确的，才能保证整个算法的正确性，对于本书中所讨论的算法，有些正确性是显而易见的，有些则对其正确性做了简单证明，其他的则将正确性证明过程省略了，因为这部分内容不属于本书所要讨论的范围。

在实践中，对于算法的正确性，我们一般采用测试的方法来验证。算法的正确性可以分为以下四个层次：①程序不含语法错误；②程序对于几组输入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、苛刻而带有刁难性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。显然，达到第④层意义下的正确是极为困难的，所有不同输入数据的数量大得惊人，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。

(2) 算法执行速度快：即依据算法编程后在计算机上运行时所消耗的时间越少越好。

(3) 算法所占用的空间较省：即依据算法编程后在计算机上运行所占的存储量较少。其中主要考虑程序运行时所需辅助存储量的多少。

(4) 算法结构简单、易写、易读、易于转换成可运行的语言编程且易于调试和修改。这里应当指出，结构简单易写易读的算法常常不一定是运行效率最高的算法，其运算工作量可能较大（如递归算法）。

2. 算法的复杂性

复杂性是指实现或运行某一算法所需资源的多少，包括时间复杂性（time complexity）、空间复杂性（space complexity）和人工复杂性（manual complexity，指编程及改错等所需人工）。从主观上讲，我们希望选用一个既不占用很多存储、运行时间又短、其他性能指标也好的算法。然而实际上往往不可能做到十全十美，一个看上去很简单的程序，其运行时间可能要比一个形式上复杂的程序慢得多；而一个运行时间较短的程序常常要占用较多的附加存储空间。因此，应针对不同情况选用不同的算法。当前由于计算机硬件的迅猛发展，扩展内存或外存的容量已经没有什么困难了。另外由于高级语言的出现，使编程、改错等也较以前容易多了，所以本书中我们研究算法的复杂性，一般主要是指时间复杂性，偶尔也讨论某些算法的空间复杂性。

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“运行工作量”的大小，只依赖于问题的规模（通常用整数量 n 表示），或者说，它是问题规模的函数。

例如，在如下所示的两个 N×N 矩阵相乘的算法中，“乘法”运算是“矩阵相乘问题”