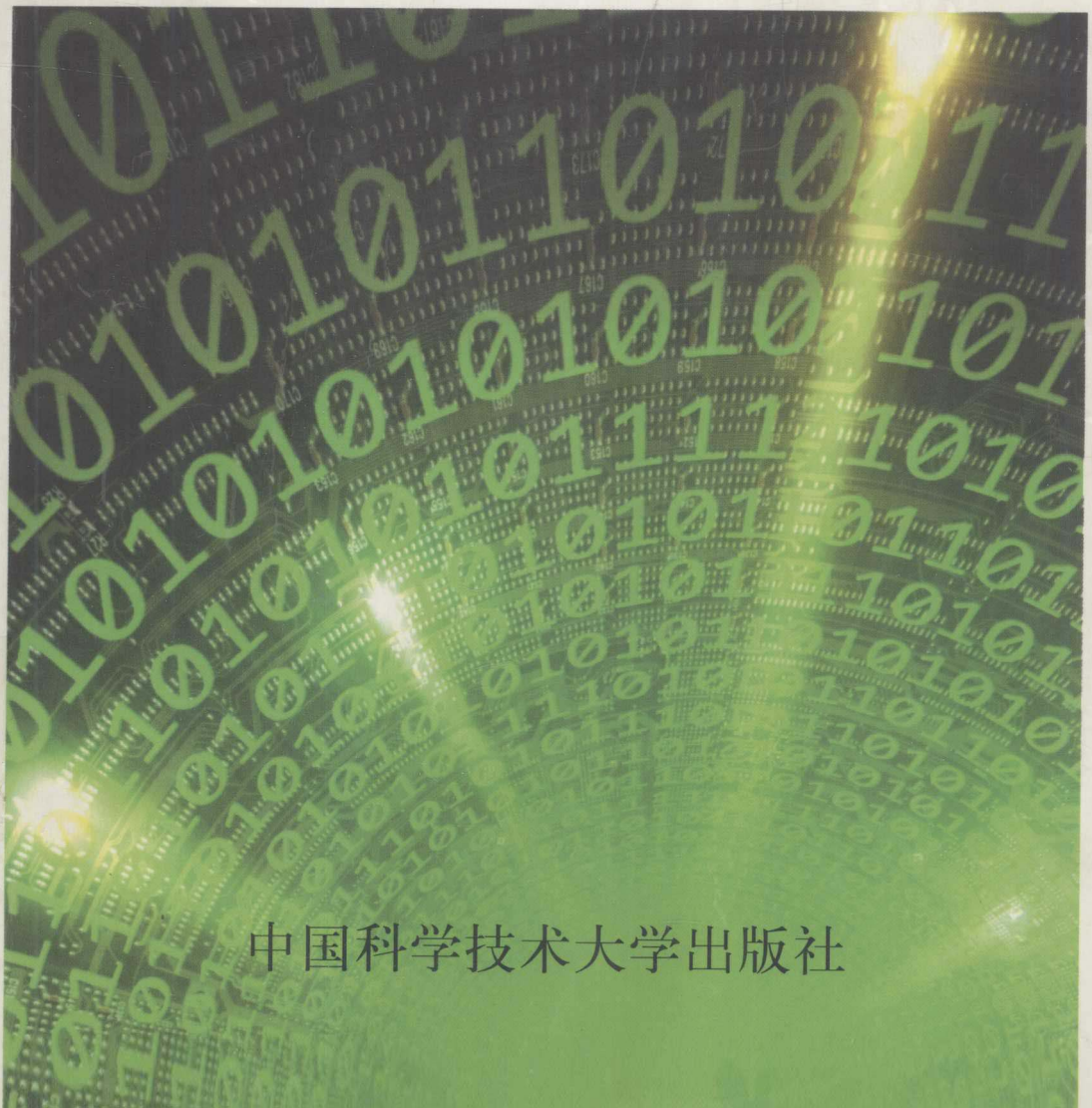
面向21世纪高等学校系列教材

数据结构

（第2版）

黄刘生　唐策善　编著



中国科学技术大学出版社

数　据　结　构

第　2　版

黄刘生　唐策善　编著

中国科学技术大学出版社
2002・合肥

内 容 简 介

本书系统地介绍各种常用的数据结构和排序、查找的各种算法。阐述了各种数据结构内在的逻辑关系、存储表示及运算操作，并对类 Pascal 语言描述的算法做了详细的注解和简要的性能分析。全书既注重原理又注重实践，配有大量图表、例题和习题，内容丰富，概念讲解清楚，逻辑性强，通俗易懂，既便于教学，又适合自学。尤其是各章的“内容提要”和“学习要点”，可以引导读者抓住重点。书中针对不同层次教学的特点和需要，用“*”号标明不同要求的区别。

本书可作为高等院校计算机类或信息类相关专业本科生、专科生及成人教育或高等职业专科院校的教材，也可供广大从事计算机软件与应用工作的科技人员及自学考试者参考。

图书在版编目(CIP)数据

数据结构/黄刘生，唐策善编著. -2 版. —合肥：中国科学技术大学出版社，2002.4
ISBN 7-312-01199-3

Ⅰ. 数... Ⅱ. ①黄...②唐... Ⅲ. 数据结构－高等学校－教材 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字(2001)第 17260 号

中国科学技术大学出版社出版发行
(安徽省合肥市金寨路 96 号，230026)
合肥义兴印务有限责任公司印刷
全国新华书店经销

开本：787mm×1092mm/16 印张：18.25 字数：500 千
1992 年 10 月第 1 版 2002 年 4 月第 2 版 2002 年 4 月第 6 次印刷
印数：35000—41000 册
ISBN 7-312-01199-3/TP・254 定价：19.80 元

第 2 版 前 言

随着个人计算机和 Internet 的迅猛发展，以计算机科学技术为核心的信息技术正在深刻地改变着人们的工作方式、生活方式和思维方式。有人预计 21 世纪的计算机软件业将成为全球高科技产业发展的最主要的推动力。因此，作为软件设计技术的理论基础，“数据结构”就不仅仅是计算机科学技术学科的核心课程，也是所有应用计算机的其它理工学科需要掌握的课程。本书是为“数据结构”课程编写的教材，其选材既满足计算机学科所需要的深度和广度，又兼顾到其它学科的需要。

本书从数据的逻辑结构、存储结构和数据的运算三个方面系统地介绍了各种最常用的数据结构。全书共分为 10 章：第 1 章到第 4 章分别介绍几种基本的线性结构，即线性表、栈、队列和串；第 5、6、7 章讨论非线性结构，它们是多维数组、广义表、树和图；第 8 章和第 9 章介绍信息处理中广泛使用的技术——排序和查找；第 10 章简要介绍外存储器上的数据结构——文件。

本书在内容表述上，既注重原理又重视实现，并配有大量解释详尽的例题，有助于读者对各种数据结构及其应用的理解和认识。对书中所涉及的有关概念及背景知识做了清楚的阐述和交代；对有关的定理及性质给出了简洁的证明；对所有算法均首先详细讨论其设计思想和方法，对一些难点算法还给出了较易理解的伪代码，然后对其逐步求精，最后给出完整的类 Pascal 语言描述，有利于读者理解算法的实质内容和基本思想。每一章的前面，列出了内容提要和学习要点，以便于读者学习和抓住重点。各章之后附有一定数量的习题（带 * 的习题难度稍大），以便读者检验学习效果和培养程序设计的能力。

本书语言流畅，通俗易懂，内容循序渐进，可以作为计算机及其它相关专业的本科或专科教材，亦可供从事计算机应用工作的科技人员和自学考试者参考。本书讲授课时为 50 至 80，教师可根据学时及专业，酌情删去带 * 的章节及 5.2 节和第 10 章等内容。

本书是笔者多年来在中国科大计算机系从事本科生的“数据结构”和研究生的“算法设计与分析”等课程教学经验的结晶，其初稿曾作为中央电视台教学用书。本书第 1 版曾得到中国科学院软件研究所所长冯玉琳教授、北京大学计算机系许卓群教授、南京大学计算系郑国梁教授，以及中国科大计算机系同事们的支持和帮助，也得到许多读者及兄弟院校讲授此教材的老师们的关切，作者谨此一并致以诚挚的谢意。

本书是黄刘生教授在第 1 版的基础上修改而成的，除了更正其中的许多错误外，对 B-树等内容进行了修改。由于水平有限，本书仍难免会有错误和缺点，殷切希望得到广大读者及同行们的批评指正。

作 者

2001 年 12 月于中国科大

目　次

第1章 概 论

[内容提要]本章的基本内容是介绍:数据和数据结构等名词和术语;描述算法的类 Pascal 语言及从时间和空间角度分析算法的方法。

[学习要点]

1. 熟悉各名词和术语的含义;掌握各种基本概念,特别是数据结构的三个方面以及这三方面的相互关系;熟悉数据结构的两大类型和四种基本的存储方法。

2. 熟悉类 Pascal 语言的书写规范,特别要注意值参和变参的区别。读者可以将类 Pascal 语言和标准的 Pascal 语言加以对比学习。

3. 了解算法的五个要素(准则)及算法与程序的区别和联系。

4. 掌握估算算法的时间复杂度的方法。

1.1 什么是数据结构

数据(Data)是信息的载体,它能够被计算机识别、存储和加工处理。它是计算机程序加工的"原料"。例如,一个代数方程求解程序中所用的数据是整数和实数,而一个编译程序或文本编辑程序中使用的数据是字符串。随着计算机软、硬件的发展,计算机应用领域的扩大,数据的含义也随之拓广了。例如,当今计算机可以处理图像、声音等。因此它们也属于数据的范畴。

数据元素(Data Element)是数据的基本单位。有些情况下,数据元素也称为元素、结点、顶点、记录。有时一个数据元素可以由若干个数据项(也可称为字段、域、属性)组成,数据项是具有独立含义的最小标识单位。

数据结构(Data Structure)指的是数据之间的相互关系,即数据的组织形式。虽然至今没有一个关于数据结构的标准定义,但它一般包括以下三方面的内容:

(1) 数据元素之间的逻辑关系,也称为数据的**逻辑结构**(Logical Structure)。

(2) 数据元素及其关系在计算机存储器内的表示,也称为数据的**存储结构**(Storage Structure)。

(3) 数据的运算,即对数据施加的操作。

数据的逻辑结构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抽象出来的数学模型。数据的存储结构是逻辑结构在计算机存储器里的实现(亦称为映象),它是依赖于计算机的,对机器语言而言,存储结构是具体的,但我们只在高级语言的层次上来讨论存储结构。数据的运算是定义在数据的逻辑结构上的,每种逻辑结构都有一个运算的集合。例如,最常用的运算有:检索,插入,删除,更新,排序等。这些运算实际上是在抽象的数据上所施加的一系列抽象的操作,所谓抽象的操作,是指我们只知道这些操作是"做什么",而无须考虑"如何做"。只有确定了存储结构之后,我们才考虑如何具体实现这些运算。本书中讨论的数据运算,均在高级语言的层次上用相应的算法来实现。

为了增加对数据结构的感性认识，下面举一例来具体说明上述概念。

例 1.1 学生成绩表为(见下页表)：

学　号	姓名	数学分析	普通物理	高等代数	平均成绩
880001	丁一	90	85	95	90
880002	马二	80	85	90	85
880003	张三	95	91	99	95
880004	李四	70	84	86	80
880005	王五	91	84	92	89
…	…	…	…	…	…

我们把这张学生成绩表称为一个数据结构，表中的每一行为一个数据元素(或结点、或记录)，它由学号、姓名、各科成绩及平均成绩等数据项组成。该表中数据元素之间的逻辑关系是：对表中任一结点，与它相邻且在它前面的结点(亦称为**直接前趋**)(Immediate Predecessor)最多只有一个；与表中任一结点相邻且在其后的结点(亦称为**直接后继**(Immediate Successor)也最多只有一个。表中只有第一个结点没有直接前趋，故称为**开始结点**，同时，也只有最后一个结点没有直接后继，故称之为**终端结点**。例如，表中"马二"所在结点的直接前趋结点和直接后继结点分别是"丁一"和"张三"所在的结点，上述结点间的关系构成了这张学生成绩表的逻辑结构。该表的存储结构则是指在计算机存储器中如何表示结点之间的这种关系，即表中的结点是顺序地邻接存储在一片连续的单元之中，还是用指针将这些结点链接在一起？在这张表中，可能要经常查看某一学生的成绩，当学生退学时要删除相应的结点，招收新生时要增加结点。究竟怎样进行查找、删除、插入，这就是数据的运算问题。搞清楚了上述三个问题，也就弄清了学生成绩表这个数据结构。

综上所述，我们可以将数据结构定义为：按某种逻辑关系组织起来的一批数据，按一定的存储表示方式把它们存储在计算机的存储器中，并在这些数据上定义了一个运算的集合，就叫做一个**数据结构**。

在不易产生混淆之处，我们常常将数据的逻辑结构简称为数据结构，数据的逻辑结构有两大类：

1. 线性结构

线性结构的逻辑特征是：若结构是非空集，则有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。线性表就是一个典型的线性结构。本书的第 2 章到第 4 章介绍的都是线性结构。

2. 非线性结构

非线性结构的逻辑特征是一个结点可能有多个直接前趋和直接后继。第 5 章到第 7 章讨论的数据结构都是非线性结构。

数据的存储结构可用以下四种基本的存储方法得到：

1) *顺序存储方法*

该方法是把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为**顺序存储结构**(Sequential Storage Structure)，通常顺序存储结构是借助于程序语言的向量来描述的。

该方法主要应用于线性的数据结构，非线性的数据结构也可以通过某种线性化的方法来

实现顺序存储。

2) 链接存储方法

该方法不要求逻辑上相邻的结点其物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。由此得到的存储表示称为**链式存储结构**(Linked Storage Structure)，通常要借助于程序语言的指针类型来描述它。

3) 索引存储方法

该方法通常是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)，关键字是能唯一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)；若一组结点在索引表中只对应一个索引项，则该索引表称之为**稀疏索引**(Sparse Index)。稠密索引中索引项的地址指示结点所在的存储位置，而稀疏索引中索引项的地址则指示一组结点的起始存储位置。

4) 散列存储方法

该方法的基本思想是根据结点的关键字直接计算出该结点的存储地址。

上述四种基本的存储方法也可以组合起来对数据结构进行存储映象。同一种逻辑结构采用不同的存储方法，可以得到不同的存储结构。选择何种存储结构来表示相应的逻辑结构，主要是使其运算方便及根据算法的时空要求来具体确定。

值得指出的是，很多教科书上是将数据的逻辑结构和数据的存储结构定义为数据结构，而将数据的运算定义为数据结构上的操作。但是，无论怎样定义数据结构，都应该将数据的逻辑结构、数据的存储结构及数据的运算这三方面看成一个整体，希望读者学习时，不要孤立地去理解一个方面，而要注意它们之间的联系。

正是因为存储结构是数据结构不可缺少的一个方面，所以我们常常将同一逻辑结构的不同存储结构冠以不同的数据结构名称来标识它们。例如，线性表是一种逻辑结构，若采用顺序方法的存储表示，则称该结构为顺序表；若采用链接方法的存储表示，则称该结构为链表；若采用散列方法的存储表示，则可称其为散列表。

同理，由于数据的运算也是数据结构不可分割的一个方面，所以给定了数据的逻辑结构和存储结构，若定义的运算集合及其运算的性质不同，也可能导致完全不同的数据结构。例如，若对线性表上的插入、删除运算限制仅在表的一端进行，则该线性表称之为栈；而插入限制在表的一端进行，删除限制在表的另一端进行的线性表称为队列。更进一步，若线性表采用顺序表和链表作为存储结构，则对插入和删除运算做了上述限制之后可分别得到顺序栈和链栈，顺序队列和链队列。

1.2 学习数据结构的意义

数据结构是计算机软件和计算机应用专业的核心课程之一。在计算机众多的系统软件和应用软件中都要用到各种数据结构。因此，仅掌握几种计算机语言难以应付众多的复杂问题，要想有效地使用计算机，还必须学习数据结构的有关知识。

在计算机发展的初期，人们使用计算机主要是处理数值计算问题，由于当时所涉及的运算对象是简单的整型、实型和布尔型数据，所以程序设计者的主要精力是集中于程序设计的技巧上，而无须重视数据结构。随着计算机应用领域的扩大和软硬件的发展，“非数值性问题”越来

越显得重要。据统计，当今处理非数值性问题占用了90%以上的机器时间，这类问题所涉及到的数据结构更为复杂，数据元素之间的相互关系一般无法用数学方程式加以描述。因此，解决此类问题的关键已不再是分析数学和计算方法，而更重要的是要能设计出合适的数据结构，才能有效地解决问题。

著名的瑞士计算机科学家沃斯(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。请看下面的两个例子。

例 1.2 电话号码查询问题。

假定要编写一个程序查询某个城市或单位的私人电话号码，要求对任意的姓名，若该人有电话，则能迅速找到其电话号码；否则指出该人没有电话。解此问题首先要构造一张电话号码登记表，表中每个结点存放两个数据项：姓名和电话号码。要写出好的查找算法，取决于这张表的结构及存储方式。最简单的方式是将表中结点顺序地存储在计算机中，查找时从头开始依次查对姓名，直到找出正确的姓名或是找遍整个表均没有找到为止。这种查找算法对于一个不大的单位或许是可行的，但对于一个有成千上万私人电话的城市就不实用了。然而，若这

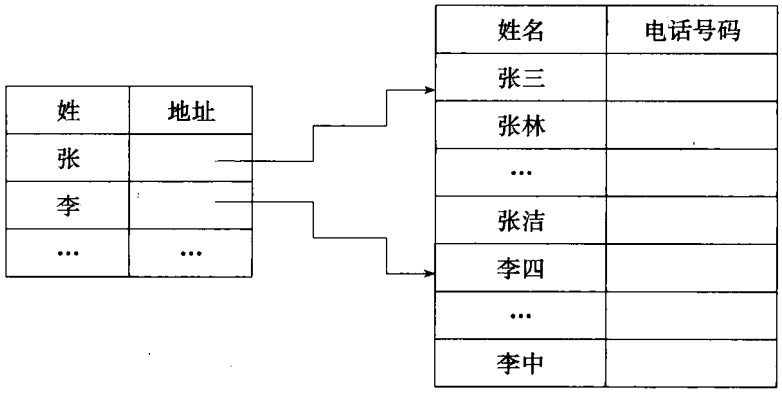


图 1.1 电话号码查询问题的索引存储

张表是按姓氏排列的，则我们可以另造一张姓氏索引表，采用如图 1.1 所示的存储结构。那么查找过程是先在索引表中查对姓氏，然后根据索引表中的地址到电话号码登记表中核查姓名，这样查找登记表时就不需查找其它姓氏的名字了。因此，在这种新的结构上产生的查找算法就更为有效。在查找一章中我们还要讨论有关的查找策略。

＊例 1.3 课程考试时间安排问题

假设计算机系某学期为研究生开设了六门专业课：算法分析，形式语言，计算机图形学，模式识别，计算机网络，人工智能。五名研究生的选课情况如图 1.2 所示。

姓　名	选修课 1	选修课 2	选修课 3
丁　一	算法分析	形式语言	计算机网络
马　二	计算机图形学	模式识别	
张　三	计算机图形学	计算机网络	人工智能
李　四	模式识别	人工智能	算法分析
王　五	形式语言	人工智能	

图 1.2 研究生选课情况表

现要求设计一个考试日程安排表，使得在尽可能短的时间内安排完考试。要能较好地解

决这一问题，首先应该选择一个合适的数据结构来表示它。为此，我们可以设计这样的一个图，图中的顶点表示课程，在所有的两门不能同时考试的课程之间连上一条边。显然同一个学生选修的几门课程是不能安排在同一时间内考试的，因此该生所选修的课程中应该两两有边相连。由此可得到如图 1.3 所示的数据结构模型，图中 A，B，…，F 分别对应上述的六门课程。例如丁一选修的三门课是 A，B，E，因此 A 和 B 之间，A 和 E 之间，B 和 E 之间均有边相连。

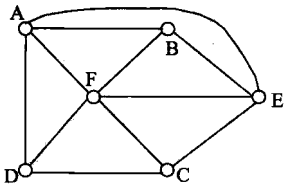


图 1.3　安排课程考试的数据结构模型

上述的由顶点和边组成的无向图是数据结构中一类非线性的数据结构，课程考试时间安排问题可以抽象为对该无向图进行“着色”操作，即用尽可能少的颜色去给图中每个顶点着色，使得任意两个相邻的顶点着上不同的颜色。每一种颜色表示一个考试时间，着上同一颜色的顶点是可以安排在同一时间内考试的课程。例如，图中顶点 A 和 C 不相邻，可选颜色 1 为它们着色；同理，B 和 D 可着同一颜色 2；E 和 F 相邻，应该分别着以不同的颜色 3 和 4。也就是说，只要安排 4 个不同的时间考试即可。时间 1 内可以考算法分析(A)和计算机图形学(C)，时间 2 内可以考形式语言(B)和模式识别(D)，时间 3 和时间 4 内分别考计算机网络(E)和人工智能(F)。

从上述例子不难看出，解决问题的一个关键步骤是选取合适的数据结构表示问题，然后才能写出有效的算法。

1.3　算法的描述

因为数据的运算是通过算法描述的，所以讨论算法是数据结构课程的重要内容之一。

通俗地讲，一个算法就是一种解题方法。更严格地说，算法是由若干条指令组成的有穷序列，它必须满足下述准则：

(1) **输入**：具有 0 个或多个输入的外界量，它们是算法开始前对算法最初给出的量。

(2) **输出**：至少产生一个输出，它们是同输入有某种关系的量。

(3) **有穷性**：每一条指令的执行次数必须是有限的。

(4) **确定性**：每条指令的含义都必须明确，无二义性。

(5) **可行性**：每条指令的执行时间都是有限的。

在一个算法中，有些指令可能是重复执行的，因此指令的执行次数可能远远大于算法中的指令条数，由有穷性可知，对于任何输入，一个算法在执行了有限条指令后一定要终止；又由可行性知道，一个算法必须在有限时间内完成。因此，一个程序如果对任何输入都不会陷入无限循环，则它就是一个算法。

算法的含义与程序十分相似，但二者是有区别的。一个程序不一定满足有穷性。例如操作系统，只要整个系统不遭破坏，它就永远不会停止，即使没有作业要处理，它仍处于一个等待循环中，以待新作业的进入。因此操作系统程序不是一个算法。另外，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。但是一个算法若用机器可执行的语言来书写，则它就是一个程序。

一个算法可以用自然语言、数学语言或约定的符号语言来描述。我们选用了一种类似于 PASCAL 的伪语言来描述算法。类 PASCAL 语言的特点是借助于 PASCAL 语言的语法结构，附之以自然语言的叙述，使得用它编写的算法既具有良好的结构，又不拘泥于具体程序语

言的某些细节。因此，类 PASCAL 语言使得算法易读易写，并且也易于转换成某种具体程序语言编写的程序。

类 PASCAL 语言的非形式化说明如下：

1）所有的算法都以如下过程或函数的形式表示：

```
procedure 过程名(形式参数表);
  begin
    语句组
  end;      //过程名
function 函数名(形式参数表)：类型名;
  begin
    语句组
  end;      //函数名
```

其中，形参表可以含有若干个值参和变参，值参和变参的选择类似于 PASCAL 语言，凡需将过程加工中的结果传递到过程外的参数必须用变参，结构类型的参数尽可能选择为变参；语句组由一个或多个语句构成，两个语句之间用分号“;”作为分隔符，不同于 PASCAL 语言的是，在过程体和函数体内增加了返回语句 **return**。正常情况下，过程体结束于 **end**，但也可以用 **return** 中途跳出过程。函数过程通常结束于语句 **return**(表达式)，并且将表达式的值返回给调用者。

2）除过程和函数中的形参表外，所有易于理解的变量说明均可以省略，并假定该语言允许一切可能出现的变量种类。

3）赋值语句的形式

变量 :=表达式;

4）条件语句有两种格式

(1)
```
if 条件 then
  语句组
endif;
```

(2)
```
if 条件 then
  语句组 1
else
  词句组 2
endif;
```

5）循环语句有四种格式

(1) “当”语句：
```
while 条件 do
  语句组
endwhile;
```

(2) “重复”语句：
```
repeat
  语句组
until 条件;
```

(3) “步进循环”语句：
```
for 变量 :=初值 to 终值 do
  语句组
endfor;
```
或
```
for 变量 :=初值 downto 终值 do
  语句组
```

```
endfor;
```

第一个 **for** 语句的工作条件是初值≤终值，步长为 1；第二个 **for** 语句的工作条件是初值≥终值，步长为−1。

6）情况语句有两种格式

(1)
```
case 表达式 of
  常量 1:语句 1;
  常量 2:语句 2;
    …
  常量 n:语句 n
endcase;
```

(2)
```
case
  条件 1:语句 1;
  条件 2:语句 2;
    …
  条件 n:语句 n
  (else 语句 n+1)
endcase;
```

其中的语句既可以是单个的语句，也可以是用 **begin** 和 **end** 括起来的复合语句。后一种 **case** 语句中的 **else** 子句是可以任选的。

7）过程调用和函数调用语句与 PASCAL 语言相同，并且也允许嵌套调用和递归调用

8）出错处理语句

```
error(字符串);
```

当过程出现异常情况时，可以使用该语句，其执行结果是通过字符串返回出错信息后，提前结束过程，直接返回到操作系统。

9）跳出循环语句

```
exit;
```

该语句终止包含它的最内层循环语句的执行，直接转移到该循环语句之后的第一条语句处。

10）读、写语句

和 PASCAL 语言类似，读语句和写语句是用两个标准过程 **read** 和 **write** 实现的，只是一般无须定义其输出格式。

11）注解的形式

```
//字符串
```

注解“//字符串”表示从“//”开始至行尾均是注释。

此外，对语句中的字符集和标识符均不加任何限制。总而言之，为了算法的简洁，在意义明确的前提下，类 PASCAL 较为自由和宽松，希望读者不要过份拘泥于语言的细节。

1.4 算法分析

求解同一个问题，可以有许多不同的算法，那么如何来评价这些算法的好坏呢？

显然，选用的算法首先应该是“正确的”。此外，主要考虑如下三点：

(1) 执行算法所耗费的时间；

(2) 执行算法所耗费的存储空间，其中主要考虑辅助存储空间；

(3) 算法应易于理解，易于编码，易于调试等等。

当然我们希望选用一个所占存储空间小、运行时间短、其它性能也好的算法。然而，实际上很难做到十全十美。原因是上述要求有时相互抵触。要节约算法的执行时间往往要以牺牲更多的空间为代价；而为了节省空间又可能要以更多的时间作代价。因此我们只能根据具体

情况有所侧重。若该程序使用次数较少，则力求算法简明易懂，易于转换为上机的程序。对于反复多次使用的程序，应尽可能选用快速的算法。若待解决的问题数据量极大，机器的存储空间较小，则相应算法主要考虑如何节省空间。本书主要讨论算法的时间特性，偶尔也讨论空间特性。

一个算法所耗费的时间，应该是该算法中每条语句的执行时间之和，而每条语句的执行时间是该语句的执行次数(也称为**频度**(Frequency Count))与该语句一次执行所需时间的乘积。但是当算法转换为程序之后，每条语句一次执行所需的时间取决于机器的指令性能、速度以及编译所产生的代码质量，这是很难确定的。因此我们假设每条语句的一次执行所需的时间均是单位时间，这样，一个算法的时间耗费就是该算法中所有语句的频度之和。于是，我们就可以独立于机器的软、硬件系统来分析算法的时间耗费。

例 1.4 求两个 n 阶方阵的乘积 C=A×B，其算法的梗概如下：

```
procedure MATRIXMLT (var A,B,C:array[1..n,1..n]of real);
  begin
(1)  for i :=1 to n do                                    n+1
(2)    for j :=1 to n do                                  n(n+1)
(3)      C[i,j] :=0;                                      n²
(4)      for k :=1 to n do                                n²(n+1)
(5)       C[i,j] :=C[i,j]+A[i,k] * B[k,j]                 n³
       endfor
     endfor
   endfor
 end;     //MATRIXMLT
```

其中右边列出的是各语句的频度。语句(1)的循环控制变量 i 要增加到 n+1，测试 i>n 成立才会终止，故它的频度是 n+1，但是它的循环体却只能执行 n 次。语句(2)作为语句(1)循环体内的语句应该执行 n 次，但语句(2)本身要执行 n+1 次，所以语句(2)的频度是 n(n+1)。同理可得到语句(3)，(4)和(5)的频度分别是 n^2，$n^2(n+1)$和 n^3。该算法中所有语句的频度之和(即算法的时间耗费)为

$$T(n)=2n^3+3n^2+2n+1 \tag{1.1}$$

由此可知，算法 MATRIXMLT 的时间耗费 T(n)是矩阵阶数 n 的函数。

一般地，我们将算法所要求解问题的输入量(或初始数据量)称为**问题的规模**(Size，**大小**)，并用一个整数表示。例如，矩阵乘积问题的规模是矩阵的阶数，而一个图论问题的规模则是图中的顶点数或边数。一个算法的**时间复杂度**(Time Complexity，也称**时间复杂性**)T(n)则是该算法的时间耗费，它是该算法所求解问题规模 n 的函数。当问题的规模 n 趋向无穷大时，我们把时间复杂度 T(n)的数量级(阶)称为**算法的渐近时间复杂度**。

例如，算法 MATRIXMLT 的时间复杂度 T(n)如(1.1)式所示，当 n 趋向无穷大时，显然有

$$\lim_{n\to\infty}\frac{T(n)}{n^3}=\lim_{n\to\infty}\frac{2n^3+3n^2+2n+1}{n^3}=2 \tag{1.2}$$

这表明，当 n 充分大时，T(n)和 n^3 之比是一个不等于零的常数，即 T(n)和 n^3 是同阶的，或说 T(n)和 n^3 的数量级相同，可记作 $T(n)=O(n^3)$。我们称 $T(n)=O(n^3)$是算法 MA-

TRIXMLT 的渐近时间复杂度。其中记号“O”是数学符号，其严格的数学定义是：

若 T(n)和 f(n)是定义在正整数集合上的两个函数，当存在两个正的常数 c 和 n_0，使得对所有的 $n \geqslant n_0$，都有 $0 \leqslant T(n) \leqslant c \cdot f(n)$成立，则 $T(n)=O(f(n))$。

当我们评价一个算法的时间性能时，主要标准是算法时间复杂度的数量级，即算法的渐近时间复杂度。例如，设有两个算法 A_1 和 A_2 求解同一问题，它们的时间复杂度分别是 $T_1(n)=100n^2$，$T_2(n)=5n^3$，当输入量 $n<20$ 时，有 $T_1(n)>T_2(n)$，后者花费的时间较少。但是，随着问题规模 n 的增大，两个算法的时间开销之比 $5n^3/100n^2=n/20$ 亦随着增大。也就是说，当问题规模校大时，算法 A_1 比算法 A_2 要有效得多，它们的渐近时间复杂度 $O(n^2)$和 $O(n^3)$ 正是从宏观上评价了这两个算法在时间方面的质量。因此，在算法分析时，往往对算法的时间复杂度和渐近时间复杂度不予区分，而经常是将渐近时间复杂度 $T(n)=O(f(n))$简称为时间复杂度，其中的 f(n)一般是算法中频度最大的语句频度。例如，算法 MATRIXMLT 的时间复杂度一般是指 $T(n)=O(n^3)$，这里的 $f(n)=n^3$ 是该算法中语句(5)的频度。

下面再举几例说明如何求算法的时间复杂度。

例 1.5

```
temp :=i;
i :=j;
j :=temp;
```

以上三条单个语句的频度均为 1，该程序段的执行时间是一个与问题规模 n 无关的常数，因此算法的时间复杂度为常数阶，记作 $T(n)=O(1)$。事实上，只要算法的执行时间不随着问题规模 n 的增长而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数，此时算法的时间复杂度也只是 O(1)。

例 1.6

```
(1)  x :=0;     y :=0;
(2)  for  k :=1  to  n  do
(3)    x :=x+1
(4)  endfor;
(5)  for  i :=1  to  n  do
(6)    for  j :=1  to  n  do
(7)      y :=y+1
(8)    endfor
(9)  endfor;
```

一般情况下，对步进循环语句只须考虑循环体中语句的执行次数，而忽略该语句中步长加 1、终值判别、控制转移等成分。因此，以上程序段中频度最大的语句是(7)，其频度为 $f(n)=n^2$，所以该程序段的时间复杂度为 $T(n)=O(n^2)$。由此可见，当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度 f(n)决定的。