数据结构

附：数据结构自学考试大纲

组编／全国高等教育自学考试指导委员会
主编／黄刘生

全国高等教育自学考试指定教材　计算机及应用专业（专科）

经济科学出

全国高等教育自学考试指定教材

计算机及应用专业（独立本科段）

数　据　结　构

（附：数据结构自学考试大纲）

全国高等教育自学考试指导委员会组编

黄刘生　主编

经济科学出版社

图书在版编目（CIP）数据

数据结构/黄刘生主编．－北京：经济科学出版社，2000.3
ISBN 7-5058-2066-4

Ⅰ.数… Ⅱ.黄… Ⅲ.数据结构 Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字（2000）第 11492 号

数 据 结 构
（附数据结构自学考试大纲）
全国高等教育自学考试指导委员会组编
黄刘生 主编
经济科学出版社出版
社址：北京海淀区万泉河路 66 号 邮编：100086
网址：www.esp.com.cn
电子邮件：esp@public2.east.net.cn
北京第二外国语学院印刷厂印刷
787×1092 16 开 15.75 印张 350000 字
2000 年 3 月第一版 2000 年 8 月第二次印刷
印数：15001—25100 册
ISBN 7-5058-2066-4/G·445 定价：21.00 元
（图书出现印装问题，请与当地教材供应部门调换）

内 容 简 介

本书系统地介绍了各种常用的数据结构以及排序、查找的各种算法。阐述了各种数据结构的逻辑关系、存储表示及运算操作。全书采用C语言作为数据结构和算法的描述语言，并对C语言描述的算法作了详细的注解和简要的性能分析。本书既注重原理又注重实践，并配有大量的图表、例题和习题，内容丰富，观点新颖，概念清楚，逻辑推理严谨，通俗易懂，既便于教学，又适合于自学。

本书可作为计算机及其应用专业独立本科段自考教材，亦可作为全日制高等院校计算机类或信息类相关专业的本科或专科教材，还可供从事计算机工程与应用工作的科技人员参考。

组编前言

当您开始阅读本书时，人类已经迈入了二十一世纪。

这是一个变幻难测的世纪，这是一个催人奋进的时代，科学技术飞速发展，知识更替日新月异。希望、困惑、机遇、挑战，随时随地都有可能出现在每一个社会成员的生活之中。抓住机遇，寻求发展，迎接挑战，适应变化的制胜法宝就是学习——依靠自己学习、终生学习。

作为我国高等教育组成部分的自学考试，其职责就是在高等教育这个水平上倡导自学、鼓励自学、帮助自学、推动自学，为每一个自学者铺就成才之路。组织编写供读者学习的教材就是履行这个职责的重要环节。毫无疑问，这种教材应当适合自学，应当有利于学习者掌握、了解新知识、新信息，有利于学习者增强创新意识、培养实践能力、形成自学能力，也有利于学习者学以致用、解决实际工作中所遇到的问题。具有如此特点的书，我们虽然沿用了“教材”这个概念，但它与那种仅供教师讲、学生听，教师不讲、学生不懂，以“教”为中心的教科书相比，已经在内容安排、形式体例、行文风格等方面都大不相同了。希望读者对此有所了解，以便从一开始就树立起依靠自己学习的坚定信念，不断探索适合自己的学习方法，充分利用已有的知识基础和实际工作经验，最大限度地发挥自己的潜能，以达到学习的目标。

欢迎读者提出意见和建议。

祝每一位读者自学成功。

全国高等教育自学考试指导委员会

1999.5

编者的话

随着个人计算机和 Internet 的飞速发展，形形色色的信息处理系统已渗入到社会和生活的各个领域。但是，信息处理系统的软件和硬件这两大组成部分发展极不平衡，与飞速发展的硬件相比，软件的发展速度相对迟缓，这就需要越来越多的人掌握设计高性能软件的技术，以推动社会信息化的进程。因为无论是系统软件还是应用软件，其核心是数据结构及其算法，所以作为软件设计技术的理论基础，“数据结构”就不仅仅是计算机学科的核心课程，也应该是所有应用计算机的其它学科所必须掌握的课程。

本书是根据全国高等教育自学考试计算机应用专业(独立本科段)的“数据结构”课程自学考试大纲编写的教材，所选内容符合自考大纲的要求。全书共十章，第 1 章介绍了数据、数据结构、抽象数据类型及算法的性能分析等基本概念；第 2 章至第 7 章分别讨论了线性表、栈、队列、串、多维数组、广义表、树和图等几种基本的数据结构及其应用；第 8、9 章两章讨论了信息处理中广泛使用的技术——排序和查找；第 10 章简要介绍了外存储器上的数据结构——文件。

本书在内容表述上，参考了目前国际上几本流行的教材，并结合笔者多年来从事“数据结构”、“算法设计与分析”及“面向对象的程序设计”等课程的教学经验和体会，介绍了一些新的观点和方法，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晰的阐述和交代；对有关的定理及性质给出了简明的证明；对所有算法，均首先详细讨论其设计思想和实现方法，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 C 代码。每章后面均附有一定数量的习题(带 * 的习题难度稍大)，可供读者练习及进一步思考。

本书力求叙述通俗易懂、内容循序渐进、示例丰富，以便于自学。除作为自学考试教材外，本书可供计算机及相关专业各类本科或专科作为教材或教学参考书，亦可供从事计算机应用工作的科技人员参考，对使用 C 语言编程者尤为适合。若使用本书作为专科教材，可酌情删去 5.2,6.4,6.5.2,6.5.3,8.2.2,9.3.2 及第 10 章等章节。

在本书编写过程中，得到了全国高等教育自学考试指导委员会——电子电工与信息类专业委员会副主任、中国科大计算机系陈国良教授，专业委员会秘书长、上海交大电子信息学院陈敏逊教授，以及中国科大计算机系领导和同事们的支持和帮助。书稿付印前，承蒙南京大学计算机系郑国梁教授、金志权教授，中国科大计算机系龚育昌教授，安徽大学计算机系谢荣传教授详细审阅，并提出了许多宝贵意见。笔者在此一并致以诚挚的谢意。

由于作者水平有限、时间仓促，书中难免会有缺点和错误，恳请广大读者及同行们批评指正。

编　者

1999 年 8 月于合肥

目　录

数据结构

数据结构自学考试大纲

第1章 概 论

自1946年首台计算机问世以来，计算机的应用已深入到国民经济的各个领域，人类已跨入了信息时代。与飞速发展的计算机硬件相比，计算机软件的发展相对缓慢。因为软件的核心是算法，所以对算法的深入研究必将促进计算机软件的发展。而算法实际上是加工数据的过程，因此，研究数据结构对设计高性能算法及高性能软件至关重要。

1.1 基本概念和术语

本节将对一些基本概念和术语加以定义和解释。

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。随着计算机软件、硬件的发展，以及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个方面的内容：

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，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用计算机语言的实现(亦称为映象)，它是依赖于计算机语言的，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，均以C语言描述的算法来实现。

数据类型(Data Type)自 20 世纪 70 年代以来,几乎所有的高级程序设计语言都提供了这一概念。所谓数据类型是一个值的集合以及在这些值上定义的一组操作的总称。例如,C 语言的"整数类型"就定义了一个整数可取值的范围(其最大值 INT - MAX 依赖于具体机器)以及对整数可施加的加、减、乘、除和取模等操作。

按"值"是否可分解,可将数据类型划分为两类:**原子类型**,其值不可分解,如 C 语言的整型、字符型等标准类型及指针等简单的导出类型;**结构类型**,其值可分解为若干个成分(或称为分量),如 C 的数组、结构等类型。原子类型通常是由语言直接提供的,而结构类型则是用户借助于语言提供的描述机制自己定义的,它通常是由标准类型派生的,故它也是一种导出类型。通常数据类型可以看作是程序设计语言中已实现的数据结构。

抽象数据类型(Abstract Data Type 简称 ADT)是指抽象数据的组织和与之相关的操作。它可以看作是数据的逻辑结构及其在逻辑结构上定义的操作。一个 ADT 可描述为:

```
ADT ADT-Name {
    Data: // 数据说明①
    数据元素之间逻辑关系的描述
    Operations: // 操作说明
        Operation1: // 操作 1,它通常可用 C 或 C++ 的函数原型来描述
            Input: 对输入数据的说明
            Preconditions: 执行本操作前系统应满足的状态 // 可看作初始条件
            Process: 对数据执行的操作
            Output: 对返回数据的说明
            Postconditions: 执行本操作后系统的状态 // "系统"可看作某个数据结构
        Operation2: // 操作 2
        ……
} // ADT
```

抽象数据类型可以看作是描述问题的模型,它独立于具体实现。它的优点是将数据和操作**封装**在一起,使得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**信息隐藏**。在 C++ 中,我们可以用类(包括模板类)的说明来表示 ADT,用类的实现来实现 ADT(参阅[10])。因此,C++ 中实现的类相当于是数据的存储结构及其在存储结构上实现的对数据的操作。

ADT 和类的概念实际上反映了程序或软件设计的两层抽象:ADT 相当于是在**概念层**(或称为抽象层)上描述问题,而类相当于是在**实现层**上描述问题。此外,C++ 中的类只是一个由用户定义的普通类型,可用它来定义变量(称为**对象**或类的实例)。因此,在 C++ 中,最终是通过操作对象来解决实际问题的,所以我们可将该层次看作是**应用层**。例如,main 程序就可看作是用户的应用程序。

---

① "// 文字序列"是 C++ 的单行注释,即从"//"开始至行尾均是注释。本书的 C 程序中亦采用此方式来写注释,它不一定适用于你所使用的 C 版本。

由于C语言中没有提供“类”这一数据类型，因此无法实现ADT，故我们不采用ADT的形式来描述数据结构，以节省篇幅。读者只要记住，它实际上等价于我们定义的数据的逻辑结构以及在逻辑结构上定义的抽象操作。

为了增加对数据结构的感性认识，下面举例来说明有关数据结构的概念。

【例1.1】学生成绩表，见表1.1。

表1.1　　学生成绩表

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
⋮	⋮	⋮	⋮	⋮	⋮

我们把表1.1称为一个数据结构，表中的每一行是一个结点（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：对表中任一个结点，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**；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用计算机语言如何表示结点之间的这种关系，即表中的结点是顺序邻接地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，进来新学生时要增加结点。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。

在不会产生混淆的前提下，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。本书第1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数组来描述的。

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在物理位置上亦相邻，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)，关键字是能惟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，则该索引表称之为**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，既可以单独使用，也可以组合起来对数据结构进行存储映像。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，视具体要求而定，主要考虑的是运算方便及算法的时空要求。

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，无论是怎样定义数据结构，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，在给定了数据的逻辑结构和存储结构之后，按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、删除运算限制在表的一端进行，则该线性表称之为栈；若对插入限制在表的一端进行，而删除限制在表的另一端进行，则该线性表称为队列。更进一步，若线性表采用顺序表或链表作为存储结构，则对插入和删除运算做了上述限制之后，可分别得到顺序栈或链栈，顺序队列或链队列。

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。因此，仅掌握几种计算机语言难以应付众多复杂的课题，

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、实型或布尔型数据，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，而是要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。对任意给出的一个姓名，若该人有电话号码，则要迅速地找到其电话号码；否则指出该人没有电话号码。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，但对一个有成千上万私人电话的城市就不实用了。然而，若这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无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。

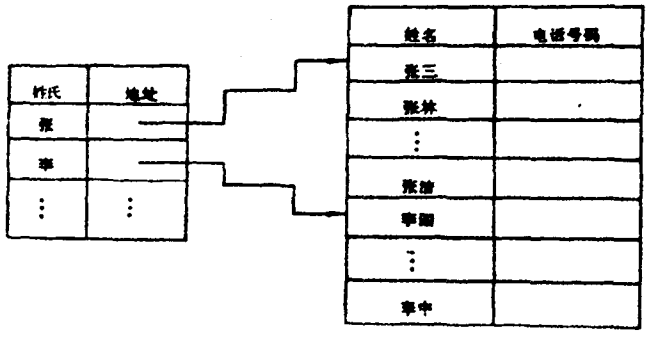

图 1.1　电话号码查询问题的索引存储

【例 1.3】田径赛的时间安排问题。

假设某校的田径选拔赛共设六个项目的比赛，即跳高、跳远、标枪、铅球、100 米和 200 米短跑，规定每个选手至多参加三个项目的比赛。现有五名选手报名参赛，选手所选择的项目如表 1.2 所示。

现在要求设计一个竞赛日程安排表，使得在尽可以短的时间内安排完比赛。为了能较好地解决这个问题，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(见图 1.2),图中顶点代表竞赛项目,在所有的两个不能同时进行比赛的项目之间连上一条边。显然同一个选手选择的几个项目是不能在同一时间内比赛的,因此该选手所选择的项目中应该两两有边相连。由此可得到如图 1.2 所示的数据结构模型,图中 A,B,…,F 分别对应于六个竞赛项目。例如,丁一选择的三个项目是 A,B 和 E,因而 A 与 B 之间,A 与 E 之间以及 B 与 E 之间均有边相连。

表 1.2　参赛选手比赛项目表

姓名	项目 1	项目 2	项目 3
丁一	跳高	跳远	100 米
马二	标枪	铅球	—
张三	标枪	100 米	200 米
李四	铅球	200 米	跳高
王五	跳远	200 米	—

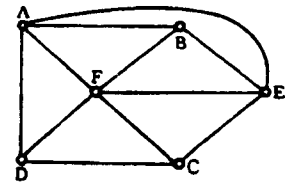

图 1.2　安排竞赛项目的数据结构模型

上述由顶点和边组成的无向图是数据结构中非线性数据结构中的一类。竞赛项目的时间安排问题可以抽象为对该无向图进行"着色"操作,即使用尽可能少的颜色去给图中每个顶点着色,使得任意两个有边连接的相邻顶点着上不同的颜色。每一种颜色表示一个比赛时间,着上同一颜色的顶点是可以安排在同一时间内竞赛的项目。例如,图中顶点 A 和 C 不相邻,可以选取颜色 1 为它们着色;同理,B 和 D 可着同一种颜色 2;E 和 F 有边相连,它们是相邻的,应该分别着上颜色 3 和颜色 4。也就是说,只要安排 4 个不同的时间竞赛即可。时间 1 内可以比赛跳高(A)和标枪(C),时间 2 内可以比赛跳远(B)和铅球(D),时间 3 和时间 4 内分别比赛 100 米(E)和 200 米(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该问题,然后才能写出有效的算法。

1.3　算法的描述和分析

因为数据的运算是通过算法(Algorithm)描述的,所以讨论算法是数据结构课程的重要内容之一。

非形式地说,算法是任意一个良定义的计算过程,它以一个或多个值作为输入,并产生一个或多个值作为输出。因此,一个算法是一系列将输入转换为输出的计算步骤。

一个算法也可以被认为是用来解决一个计算问题的工具。计算问题的陈述指定了所渴望的输入输出之间的关系,而相应的算法则描述了达到这种输入输出关系的计算过程。例如,假设我们考虑的计算问题是这样一个排序问题:将一个数字序列排序为非降序,该问题的形式定义由满足下述关系的输入输出序列构成:

输入:数字序列$\langle a_1, a_2, \cdots, a_n \rangle$。

输出:输入序列的一个枚举$\langle a_1', a_2', \cdots, a_n' \rangle$使得 $a_1' \leqslant a_2' \leqslant \cdots \leqslant a_n'$

对于一个具体的输入序列〈31,41,59,26,41,58〉,排序算法应该返回的输出序列是〈26,31,41,41,58,59〉。这样的一个输入序列称为该排序问题的一个输入实例。一般地,一个问题的**输入实例**是满足问题陈述中所给出的限制、为计算该问题的解所需要的所有输入构成的。

若一个算法对于每个输入实例均能终止并给出正确的结果,则称该算法是正确的。正确的算法解决了给定的计算问题。一个不正确的算法是指对某些输入实例不终止,或者虽然终止但给出的结果不是所渴望得到的答案,一般地说,我们只考虑正确的算法。

一个算法可以用自然语言、计算机程序语言或其它语言来说明,惟一的要求是该说明必须精确地描述计算过程。一般而言,描述算法最合适的语言是介于自然语言和程序语言之间的伪语言,它的控制结构往往类似于 Pascal、C 等程序语言,但其中可使用任何表达能力强的方法使算法表达更加清晰和简洁,而不至于陷入具体的程序语言的某些细节。但是,考虑到读者易于上机验证算法和提高读者的实际程序设计能力,本书将采用 C 语言描述算法。由于 C 语言版本众多,本书将尽可能使用美国国家标准学会颁布的 ANSI C。

为方便起见,这里先定义一个错误处理函数,它输出错误信息后退出程序运行。该函数将在后续章节的许多程序中用来简化错误处理代码。

```
#include <stdlib.h>        //其中有 exit 的说明
#include <stdio.h>         //其中有标准错误 stderr 的说明
void Error(char * message)
{
    fprintf(stderr,"Error:%s\n",message);//输出错误信息
    exit(1);//终止程序,返回 1 给操作系统
}
```

求解同一计算问题可能有许多不同的算法,究竟如何来评价这些算法的好坏以便从中选出较好的算法呢?

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;而为了节省空间可能要耗费更多的计算时间。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;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))与该语句执行一次所需时间的乘积。但是,当算法转换为程序之后,每条语句执行一次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。我们假设每条语句执行一次所需的时间均是单