中等职业学校电子信息类教材（计算机技术专业）

数据结构（C 语言版）

朱若愚　张秋璞　等编著

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

内 容 简 介

数据结构是计算机程序设计的重要理论技术基础课程。本书作为面向中等职业学校的计算机相关专业学生而编写的教材，系统地介绍了数据结构的基础知识和应用。

本书从逻辑结构和物理存储结构两个方面详细介绍了数据存储的原理和方法，根据各种存储结构的特点介绍了大量常用的计算方法。对于结构相对简单而又常见的线性结构，本书列举了许多实例，力求全面、细致地体现线性结构的优点和缺点。为了使线性链表这种比较抽象的数据结构易于被理解，本书的例题都力求具体化，还添加了形象的插图。在介绍树和图等非线性数据结构时，在阐明概念的基础上增加了实用性和趣味性较强的例子。在介绍排序和查找等研究计算方法的内容时，不但给出了用类 C 语言表示的算法，对于较复杂的算法还给出了源程序。本书还对变量含义、设计思想和使用到的其他数据结构进行了详细说明。此外，还对许多算法执行过程中数据的变化进行了说明，体现了“化复杂为简单、化抽象为具体”的教学原则。

考虑到中职学生的年龄和知识结构特点，本书特别注重内容的基础性和典型性。因此，也可以作为其他有志于从事程序设计的青少年或计算机爱好者的参考书。

图书在版编目（CIP）数据

数据结构：C 语言版/朱若愚，张秋璞等编著. —北京：电子工业出版社，2004.9
中等职业学校电子信息类教材·计算机技术专业
ISBN 7-5053-9932-2

Ⅰ. 数… Ⅱ. ①朱…②张… Ⅲ. ①数据结构—专业学校—教材②C 语言—程序设计—专业学校—教材
Ⅳ. ①TP311.12②TP312

中国版本图书馆 CIP 数据核字（2004）第 091314 号

责任编辑：李 玮
特约编辑：刘 嘉
印 刷：
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
经 销：各地新华书店
开 本：787×1092 1/16 印张：11.75 字数：300.8 千字
印 次：2004 年 9 月第 1 次印刷
印 数：5 000 册 定价：15 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。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。

前　言

计算机作为信息处理的重要工具，已经应用到当今生活的各个方面，成为人们工作、学习和生活离不开的好帮手。在它从诞生到现在经历的半个多世纪中，以硬件为基础的软件在程序规模、指令功能、编程方法及软件易用性等方面发生了巨大的变化。作为软件发展的最基本和最重要的方式，虽然一些辅助编程系统已经能自动生成一些程序代码，但程序设计的主要工作还是由程序设计人员完成。本书所介绍的数据结构正是研究程序设计最重要的专业基础课之一。

计算机程序的主要任务是处理数据，这就使得大量的数据在计算机中怎样存储成为一个重要且基本的问题。什么样的存储结构节省存储空间？什么样的存储结构有利于提高处理效率？哪一类问题适合于用哪一类存储结构？哪一种存储结构适合于哪一种算法？算法的计算复杂性如何？存储结构占用多少内存？……这一系列的问题都属于这门课程研究的内容。这里所说的存储结构包括在内存中数据的物理存储结构和体现数据之间逻辑关系的逻辑结构。而这里所说的算法是解决某个具体问题的处理过程（多以函数形式来体现）。研究算法将使我们的思路更开阔，技巧更灵活，大大有利于程序设计水平的提高。

本书作为 C 语言的后继课程，考虑到读者所掌握的基础知识和读者的年龄特点，采用了接近于 C 语言源程序的形式给出算法，对于一些较难的算法，本书直接给出了 C 语言源程序。这也是针对中职学生知识层次和知识结构特点所做的新尝试。

本书在引入基本概念和基本结构时力求简单直观，而在介绍程序结构比较复杂或综合运用前面基本数据结构的章节中则详细地分析了算法和程序特点，以帮助读者突破难点。本次编写，分析并比较了其他版本的同类教材，本着降低难度、直观易学的宗旨，增加了一些基础习题和例题，相信这将有利于初学者更好地掌握数据结构的基本知识。

朱若愚、张秋璞、胡小琳、霍莹参与了本书的编写。另外，在图版处理上，张军明、祖关涛、汪文翔、李献红也做了大量工作，在此一并致谢。

对于本书中可能存在的遗漏或不妥之处，希望读者热心指出，以使我们改进工作，提高水平。

为方便教师教学，本书还配有教学指南、电子教案和习题答案（电子版），请有此需要的教师与电子工业出版社联系，我们将免费提供（E-mail: ve@phei.com.cn; http://www.hxedu.com.cn）。

编　者

2004 年 4 月

目　录

第 1 章　绪论 …… (1)
1.1　什么是数据结构 …… (1)
1.1.1　发展历史 …… (1)
1.1.2　数据结构 …… (1)
1.2　基本概念和术语 …… (3)
1.3　算法的描述和算法分析 …… (4)
1.3.1　算法的描述 …… (4)
1.3.2　算法的分析 …… (6)
习题 1 …… (8)
第 2 章　线性表 …… (9)
2.1　线性表及其基本运算 …… (9)
2.2　线性表的存储结构 …… (11)
2.2.1　线性表的顺序存储结构 …… (11)
2.2.2　线性表的链式存储结构 …… (14)
习题 2 …… (35)
第 3 章　栈和队列 …… (36)
3.1　栈 …… (36)
3.1.1　栈的定义及其基本运算 …… (36)
3.1.2　栈的存储结构 …… (36)
3.2　队列 …… (48)
3.2.1　队列的定义及基本运算 …… (48)
3.2.2　队列的顺序存储结构 …… (49)
3.2.3　队列的链式存储结构——链队列 …… (54)
习题 3 …… (56)
第 4 章　串和数组 …… (57)
4.1　串的运算 …… (57)
4.1.1　字符串的概念 …… (57)
4.1.2　字符串的运算 …… (57)
4.2　串的存储结构 …… (60)
4.2.1　串的顺序存储 …… (60)
4.2.2　串的链式存储 …… (62)
4.2.3　两种存储方式存储串时的操作 …… (63)

4.3 数组 (64)
4.3.1 有关数组（Array）的一些概念 (64)
4.3.2 数组存储的排列顺序 (65)
4.3.3 数组的运算 (65)
4.3.4 数组的顺序存储 (66)
4.4 稀疏矩阵 (66)
4.4.1 稀疏矩阵 (66)
4.3.2 稀疏矩阵的存储 (68)
4.3.3 稀疏矩阵的运算 (69)
习题 4 (73)
第 5 章 树 (74)
5.1 树的定义和运算 (74)
5.1.1 树的定义 (74)
5.1.2 树的基本术语 (76)
5.1.3 树的基本运算 (76)
5.2 二叉树的遍历 (76)
5.2.1 二叉树的定义 (76)
5.2.2 二叉树的遍历 (77)
5.3 二叉排序树 (80)
5.3.1 二叉排序树的定义 (80)
5.3.2 二叉排序树的插入、生成和删除 (80)
5.4 哈夫曼树 (83)
5.4.1 基本术语 (83)
5.4.2 构造哈夫曼树 (85)
5.4.3 哈夫曼树的应用 (86)
习题 5 (91)
第 6 章 查找 (92)
6.1 顺序查找 (92)
6.2 折半查找 (95)
6.3 分段查找 (97)
6.4 树形结构的查找 (99)
6.4.1 汉字内码的查找 (99)
6.4.2 利用二叉排序树查找 (100)
6.5 哈希表的查找 (102)
6.5.1 哈希查找的有关概念 (102)
6.5.2 构造哈希函数的常用方法 (102)
6.5.3 处理地址冲突的常用方法 (105)
6.5.4 哈希查找算法的评价 (106)

习题 6 …… (107)
第 7 章 排序 …… (108)
7.1 排序的有关概念和术语 …… (108)
7.2 选择排序 …… (109)
7.3 堆排序 …… (114)
7.4 起泡排序 …… (118)
7.5 插入排序 …… (121)
7.6 移动最少和比较最少的插入排序 …… (127)
7.6.1 链表插入排序 …… (127)
7.6.2 折半插入 …… (128)
7.6.3 希尔分类 …… (130)
7.7 利用二叉树进行插入排序 …… (133)
7.8 快速分类 …… (136)
7.9 合并排序与外部排序 …… (139)
7.9.1 合并排序 …… (139)
7.9.2 外部排序 …… (142)
7.10 多关键字排序 …… (142)
7.10.1 低关键字优先排序 …… (143)
7.10.2 高关键字优先排序 …… (145)
习题 7 …… (148)
第 8 章 图 …… (150)
8.1 图的概念及术语 …… (150)
8.2 图的存储结构 …… (151)
8.2.1 多重表 …… (151)
8.2.2 邻接矩阵 …… (152)
8.2.3 关联矩阵 …… (152)
8.2.4 邻接表 …… (153)
8.2.5 十字链表 …… (155)
8.3 图的深度优先搜索 …… (155)
8.4 广度优先搜索 …… (161)
8.5 图的连通性 …… (167)
8.6 最短路问题 …… (173)
习题 8 …… (178)

第1章 绪 论

本章将结合数据结构的发展史，介绍数据结构的基本概念。为了方便读者学习后面的知识，这一章将介绍必要的基础知识，包括数据结构的基本定义和术语。同时，还将介绍类 C 语言的各种句型及算法描述的规范。本章介绍的实例源自生活，通俗易懂，有助于读者理解数据结构中的抽象概念。

1.1 什么是数据结构

1.1.1 发展历史

自 1946 年电子计算机问世以来，电子计算机的应用范围越来越广泛。计算机科学和软、硬件技术发展迅速。计算机的应用领域也从早期的科学计算和工程计算扩展到人类社会活动的更多领域。计算机的处理对象已经不仅是简单的纯数值型信息，而是带有不同结构的各种数据和非数值型信息。现代计算机科学把计算机程序处理的一切数值型和非数值型的信息统称为数据（Data），电子计算机只是作为加工处理数据的工具。因此，要设计一个好的软件，不仅要求人们掌握计算机语言，还要求人们系统地研究计算机程序加工的对象，即研究数据的特性和数据之间存在的关系——数据结构。

数据结构作为一门独立的课程从 1968 年就在美国被设立了。20 世纪 60 年代中期数据结构课程的前身称做表处理语言，但当时没有形成课程体系。在 1968 年，美国的唐·欧·克努特教授开创了数据结构的最初体系，并且系统地阐述了数据的逻辑结构和存储结构。在美国一些大学的计算机系，也开始把数据结构作为一门独立的课程。从 20 世纪 60 年代到 20 世纪 70 年代初，人们越来越重视数据结构。20 世纪 70 年代中期到 20 世纪 80 年代初，关于数据结构的书籍开始大量出现，我国各种关于计算机的学术研讨会也将数据结构确定为计算机专业的主要课程，而且也是非计算机专业学生的主要选修课程。这是因为在计算机科学中，数据不仅是一般程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。

数据结构的发展并未终结。一方面，它向研究各个专门领域中特殊问题的数据结构发展；另一方面，人们从抽象数据类型的观点来讨论数据结构已成为一种新趋势，越来越受到重视。

1.1.2 数据结构

用计算机可以解决图书馆的书目检索自动化、交通灯的管理及成绩排队等问题。对这些

问题的解决需要从具体问题抽象出一个恰当的数学模型，并且设计一个较好的解决此数学模型的算法，编写出源程序并进行测试，通过调整得到最终程序。由此可以看出，用计算机解决实际问题的主要步骤是寻求数学模型，而寻求数学模型的实质是分析问题，从中提取操作的对象，并且找出这些操作对象之间含有的关系，然后用数学的语言加以描述。至于什么是数据结构，我们不妨先看下面的例子，然后再给出定义。

【例 1.1】 在我们熟悉的 Windows 98 操作系统中，开始菜单里面有一个“文档”功能，我们打开过的文档都被它记录在案。如果使用者需要重新打开最近刚打开过的某一个文档，只要打开此功能菜单，在其中选择相应文档即可。在这个存储结构中，每打开一个文档，操作系统都会在此存储结构中记录这个文档，再打开一个文档，此结构又会接着记录新打开的文档。这种一个数据元素接一个数据元素的存储方式就是我们即将学习的线性存储结构。随着打开文档数量的增加，存储结构会占用越来越多的存储空间，为节省存储空间可以设定这个存储结构的大小。例如 Windows 98 对打开文档记录数的限制是 15 个文档时，打开第 16 个文档将意味着什么呢？从实效性角度来说，保存最新打开的第 16 个文档比保存此结构中现存的第一个文档更有价值，所以 Windows 98 将会去掉其中时间最早的文档记录，而存入新打开的文档记录。这里记录新文档的操作，称为插入；而去掉时间最早的文档记录的操作称为删除。在后面的学习中会看到此结构正是线性结构中的队列。需要说明的是，我们所看到的文档列表并不一定是按队列的顺序显示，它可以按其他顺序（例如字母顺序）显示。

【例 1.2】 通讯录查询问题：假定有一个职工通讯录，记录了某个单位全体职工的姓名及相应的住址和电话。现要编写一个算法，要求给定任何一位职工的姓名，该算法能够查出该职工的住址和电话。

① 首先，确定一个适当的存储结构。有关职工的信息存储方式应服务于对这批数据的处理方式和使用目的。事实上，这样的一批数据既可以用上面例题中提到过的线性结构来存储，也可以使用我们将要学到的非线性结构（例如树）来存储。例如，我们可以把每个职工的信息看做一个记录，把这些记录一个接一个地排成普通的线性存储结构，也可以把这些记录按姓名的顺序存入数组。当然，这仍然是一个线性结构。

② 如果使用普通线性结构来存储这个通讯录，那么可以根据输入的待查职工姓名在此线性结构中一个接一个地进行比对。一旦查到便可输出该职工的住址和电话，至此查询成功。若依次查遍整个线性结构都没有找到相应的姓名，则查询失败。此种算法就是下面将要学到的顺序查找算法。

如果是对按姓名排好顺序并依次存入数组的线性结构进行查找，可以采用查询效率较高的折半查找算法。假设待查的姓名是 X，它与数组中 $a[i]$记录的姓名域比较，若相同则查找成功，否则有两种可能：若 X 小于 $a[i]$的姓名域，则只需在 $a[i]$之前继续查找；若 X 大于 $a[i]$的姓名域，则只需在 $a[i]$之后继续查找。无论如何查找区域都会缩小。对 n 名职工的数组进行查询一般不会超过 $\log_2 n+1$C 次。从上面的分析不难看出，计算机的算法与数据结构（数据模型）密切相关，数据结构直接关系到算法的选择和效率。此外，数据结构还需要给出每种结构类型所定义的各种运算的算法。仍以职工通讯录为例，每当单位增加新职工就需要添加新职工的姓名、住址和电话。这就需要在已有的结构上进行插入（Insert）。反之，有职工调离，就需要把此职工的相关信息去掉，这就是进行删除（Delete）。但是，如何实现插入和删除操作？在插入和删除后对原有数据是否有影响？有什么样的影响？如果添加职工的姓

名、住址和电话，具体应添加在什么位置？当职工调离时，删除其姓名、住址和电话，其他数据是否需要移动及怎样移动？诸如上面这类问题我们都应事先考虑。为此，需要在线性结构中定义一些运算，例如插入运算和删除运算。另外，对于职工搬家或改名字或改电话的情况还可以定义修改运算。计算机要完成这些运算，就要设计出相应的算法。为此，数据结构需给出每种结构类型所定义的各种运算的具体步骤，甚至要给出由指令序列组成的函数。

③ 最后，编写程序，进行测试、调整和修改，直至最终程序运行通过，问题得到解决。

因此，数据结构研究的主要内容是数据元素（Data element）之间抽象化的相应关系和这种关系在计算机中的存储表示，即所谓数据的逻辑结构和物理结构。它对这种结构定义相应的运算，设计出相应的算法，并且确保经过这些运算后所得到的新结构仍然是原来的结构类型。

1.2 基本概念和术语

在本节中，我们将对本书中最常用的一些概念和术语进行介绍，以便读者更好地理解以后的内容。

数据（Data）是对客观事物的符号表示，在计算机科学中是指所有输入到计算机中并被计算机程序处理的符号。例如，一本书、一篇文章、一张图表、一个句子、一个算式、一个数值、一个字符、一段影像及一段声音等都是数据。因此，数据的含义很广泛，所有能够被计算机加工的对象，能够输入计算机并能由它存储、处理和输出的对象都称为数据。

数据元素（Data element）是数据的基本单位，在计算机程序中通常作为一个对象进行考虑和处理。例如第 1.1 节里面例 1.2 中的表被称为一个数据元素。一个数据元素可能由若干个数据项组成。例如，在通讯录查询中的姓名、住址和电话就是数据项。对于一个字符串而言，每个字符就是它的数据元素。数据和数据元素是相对而言的，例如对于一个文件的每个记录而言，它相对于所在的文件是数据元素，而相对于记录中的各个数据项（域），它又是完整的数据。

数据对象（Data object）是性质相同的数据元素的集合，是数据的一个子集。例如，字符中的字母数据对象是集合 C＝{ ‘A’, ‘B’, …, ‘Z’ }。

数据结构（Data structure）是相互之间存在一种或多种特定关系的数据元素的集合。数据元素都不是孤立存在的，它们之间存在着某种关系，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，通常有四类基本结构，即集合、线性结构、树形结构和图状结构或网状结构。

数据类型（Data type）是和数据结构密切相关的一个概念，它最早出现在高级程序语言中，用以刻画程序中操作对象的特性。在用高级程序语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。

与数据结构密切相关的是定义在数据结构上的一组操作。操作的种类是没有限制的，可以根据需要进行定义，基本的操作如下所述。

① 插入（Insert）：在数据结构中的指定集合上添加新的数据元素。

② 删除（Delete）：删去数据结构中某个指定的数据元素。

③ 更新（Update）：改变数据结构中某个数据元素的值，在概念上等价于删除和插入操

作的组合。

④ 查找（Search）：在数据结构中寻找满足某个特定要求的数据元素的位置和值。

⑤ 排序（Sort）：（在线性结构中）重新安排数据元素之间的逻辑顺序关系，使之按值由小到大或由大到小的次序排列。

算法（Algorithm）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。一个算法具有以下五个主要特性。

① 有穷性：一个算法必须总是（对任何合法的输入值）在执行有穷步之后结束，而且每一步都可在有穷时间内完成。

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，并且在任何条件下，算法只有惟一的一条执行路径，即对于相同的输入只能得出相同的输出。

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

④ 输入：一个算法有零、一个或多个的输入，这些输入来自于特定的对象的集合。

⑤ 输出：一个算法有一个或多个的输出，这些输出是同输入有某个特定关系的量。

1.3 算法的描述和算法分析

1.3.1 算法的描述

算法需要用某种方式来表达，例如用人们习惯的自然语言、某种计算机语言或流程图等方式来表达。研究数据结构的目的在于有效地进行程序设计，所以在讨论各种数据结构的基本运算时都将给出相应的算法。算法需要用一种语言来描述，而描述算法的语言有很多种。在计算机上运行的程序（算法）必须严格按照语法规定用机器语言、汇编语言或高级程序语言来编写，而一个供读者阅读和交流的算法应该是描述简单、明确、令人一目了然和便于理解及掌握其思想和实质的算法。因此，本书采用类 C 语言作为算法描述语言。类 C 语言是我们在 C 语言的基础上修改而成的，它忽略了 C 语言中语法规则的一些细节，同时增强了语言描述功能。由于本课程是 C 语言的后继课程，所以本书中采用了十分接近于 C 语言的类 C 语言描述。考虑到本书读者的特点，书中对一些较为复杂的例题给出了 C 语言的准确描述（源程序）。这样做的目的是为了使算法更具体、更易于在计算机上实现。

本书中算法以函数形式表示，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
```

一般函数中使用的变量可以省略类型定义。通常，a, b, c, d, e 等用做数据元素名；i, j, k, l, m, n 等用做整型变量名；p, q, r 等用做指针变量名。

常用语句用以下形式描述。

① 预定义常量和类型如下所述。

```
//函数结果状态代码
#define   TRUE            1
#define   FALSE           0
#define   OK              1
#define   ERROR           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（存储结构）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定义。

③ 赋值语句。

```
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 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;
```

④ 选择语句。

```
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 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 n: 语句序列 n: break;
                    }default: 语句序列 n+1;
```

⑤ 循环语句。

```
for 语句          for(赋初值表达式序列; 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);
```

⑥ 结束语句。

```
函数结束语句      return 表达式;
                  return;
结束语句          break;
异常结束语句      exit(异常代码);
```

⑦ 输入和输出语句。

```
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);
```

⑧ 注释。

```
单行注释    //文字序列
```

⑨ 基本函数。

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)
求不足整数值	floor	(表达式)
进位求整数值	ceil	(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

⑩ 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。
```

1.3.2 算法的分析

在解决同一个问题时，往往能够编写出不同的算法，如何选择一个好的算法及怎样评价算法的好坏也是算法分析要讨论的问题。一般从四个方面对算法进行评价，即正确性、运行时间、占用的存储空间、是否简单。

算法正确性是设计和评价一个算法的首要条件，如果算法不正确，其他方面就无从谈起。对于算法的正确性可分为以下四个层次：

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、苛刻且带有刁难性的几组输入、输出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够产生满足规格说明要求的结果。

通常以第三层意义的正确性作为衡量一个程序是否合格的标准。

运行时间是指一个算法在计算机上运行所花费的时间。它通常等于所有语句执行时间的总和，每一条语句的执行时间总和为该语句的执行次数与单次执行所需时间的乘积，而这有时是很难精确计算的，只能在对执行次数粗略估计后计算。不管一个算法是简单还是复杂的，最终都将被分解成有确定执行时间的简单操作来具体执行。

一个算法在计算机存储器上所占用的存储空间包括存储算法本身所占用的存储空间、算法的输入、输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间三个部分。存储量需求指算法执行过程中所需要的最大存储空间。

一般来说，如果一个算法的程序描述比较简单，则它出现逻辑错误的可能性较小，它的正确性也就得到更多的保证，同时也便于编写、阅读和调试。但是这样的算法往往占用过多的时间和空间，因此效率往往不高。

在四个评价指标中，时间的减少和空间的缩小往往是矛盾的，通常用算法的执行时间作为算法好坏的主要衡量指标。这是因为硬件发展较快，存储芯片价格不断下降，使存储空间花费不断下降。通俗地说，好的算法效率高，效率高指的是算法执行时间短。对于同一个问题如果有多个算法可以解决，执行时间最短的算法效率最高。

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 $g(n)$，算法的时间量度记做 $T(n)=O(g(n))$。它表示随问题规模 n 的增大算法执行时间的增长率和 $g(n)$的增长率相同，称为算法的时间复杂度。常见的 $O(g(n))$有 $O(1)$, $O(\log_2 n)$, $O(n)$, $O(n \log_2 n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$（$O(1)$称为常量阶；$O(n)$称为线性阶；$O(\log_2 n)$称为对数阶；$O(n^2)$称为平方阶；$O(2^n)$称为指数阶）,它们的级别一个比一个高。$g(n)$中级别最高的项的级别就是 $g(n)$的级别。例如，在 $g(n)=3n^2+4n\log_2 n-100$ 中，$3n^2$ 是 $O(n^2)$阶（级别）的，$4n\log_2 n$ 是 $O(n\log_2 n)$阶的，100 是 $O(1)$阶的，所以 $g(n)$是 $O(n^2)$阶的。从这个例子中可以看出，$T(n)=O(n^2)$的含义是执行次数不超过 n^2 的某个常数倍，其他阶依次类推。一般认为，随 n 的增大，$T(n)$增长较慢的算法为优。通常说来，时间复杂度为 $O(1)$的算法不常见，一般我们尽可能选用多项式阶 $O(n)$的算法，而不希望用指数阶 $O(2^n)$的算法，即尽量选用时间复杂度级别较低的算法。

【例 1.3】　确定下列三个程序段各个语句的执行次数。

```
(1) x=x+1;
(2) for(i=1; i <= n；++ i)
(3) for(i=1; i <= n；++ i)
        for(j=1; j<= n；++ j)
```

每段的执行次数如下

段号	次数
(1)	1

(2) n

(3) n^2

值得注意的是，算法分析主要针对复杂的问题，对于比较简单或 n 很小的问题，一般无分析的必要。

习题 1

1. **现要解决图书馆的书目检索自动化问题，试用文字叙述相应算法。**
2. **试确定下面两个程序段中各个语句的执行次数。**

```
(1) for(i=1;i <= n;++i) {
        k=k+1;
    for(j=1 ; j <= n; ++ j)
            k=k+1;
    }

(2)  i=1;
     while(i<=n);
         i=i+1;
         k=k+1;
     end;

(3) for (i=1; i <= n；i=i+2) {
        k=k+1;
     for(j=1; j <= n>; j=j*2）
          k=k+1;
     }
```

第 2 章　线　性　表

计算机程序处理的数据纷繁复杂，但对于某一个具体的程序所面对的数据元素又有着相同的特性，各个数据元素之间存在着一定的逻辑关系。例如一张学生成绩单，它就是一组数据，每位同学的姓名和成绩组成一个数据元素，每个数据元素之间存在着以成绩递减为序的逻辑关系。由于数据元素间逻辑关系及其在计算机内的表示不同，就形成了不同类型的数据结构。本章讨论数据元素间具有线性关系的数据结构——线性表。

2.1　线性表及其基本运算

我们先观察下列数据结构：

【例 2.1】　某学生初中三年六学期数学成绩分别为：

表 2.1　学生数学成绩表

第 1 学期	第 2 学期	第 3 学期	第 4 学期	第 5 学期	第 6 学期
80	75	77	85	87	93

【例 2.2】　某班某天课程表为：

表 2.2　课程表

第 1 节	第 2 节	第 3 节	第 4 节	第 5 节	第 6 节
语文	数学	历史	外语	音乐	地理

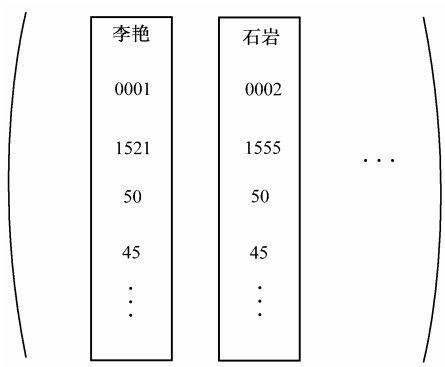
【例 2.3】　某单位职工工资表见表 2.3。

表 2.3　职工工资表

姓　名	代　号	基本工资	交通费	洗理费	…
李艳	0001	1521	50	45	…
石岩	0002	1555	50	45	…

例 2.3 稍微复杂一些，其中每一个数据元素由若干个数据项组成，常被称为一个记录，所有记录组成的工资表就是一个数据库。为了与例 2.1 和例 2.2 统一起来，例 2.3 的工资表也

表 2.4　职工工资表



可以表示为表 2.4 的形式。

像这种由几个数据元素按照一定的顺序组成的有限序列（a_0，a_1，…，a_{n-1}）被称为一个线性表。这好比用一根线穿起一串珍珠，一个数据元素为一个珍珠，顺序为穿珍珠的线。其中 a_i（i=0，1，…，n-1）可代表任意计算机程序处理的符号，例如，表 2.1 中的数据元素 a_i 为数字；表 2.2 中的数据元素 a_i 为汉语词组；表 2.4 中的数据元素 a_i 为记录。尽管如此，同一个线性表中的数据元素必定具有相同的特性，即为同一个数据对象。元素与元素之间有先后顺序关系（即数据之间的逻辑关系）。其中，a_0 为线性表的第一个数据元素；a_{n-1} 是线性表的最后一个数据元素。数据元素 a_{i-1} 在数据元素 a_i 之前并与 a_i 相邻，数据元素 a_{i+1} 在数据元素 a_i 之后并与 a_i 相邻。我们称数据元素 a_{i-1} 为数据元素 a_i 的直接前驱元素；数据元素 a_{i+1} 是数据元素 a_i 的直接后继元素。在线性表（a_0，a_1，…，a_{n-1}）中，数据元素 a_i（i =1，…，n-1）有且仅有一个直接前驱元素 a_{i-1}；数据元素 a_i（i =0，1，…，n-2）有且仅有一个直接后继元素 a_{i+1}。线性表中数据元素的个数 n（$n \geq 0$）称为线性表的长度。当 n=0 时称线性表为空表。例如，例 2.1 中“80”是线性表的第一个数据元素，它的直接后继是数据元素“75”，数据元素“93”是线性表的最后一个数据元素，它的直接前驱元素是“87”，线性表的长度是 6。

可见，线性表这种数据结构是通过数据元素在表中出现的先后顺序来体现元素间的逻辑关系的。

我们可以用如下形式来描述线性表。

含有 n 个数据元素的线性表是这样一个数据结构

$$\text{Linear\_list}=(D,\ R) \tag{2.1}$$

式中，$D=\{a_i \mid a_i \in \text{data object},\ 0 \leq i \leq n-1,\ n \geq 0\}$

$R=\{< a_{i-1}, a_i > \mid a_{i-1}, a_i \in d,\ 1 \leq i \leq n-1\}$

通过上面的分析，我们对线性表这种数据结构有了一定的了解，下面介绍线性表的基本运算。

① 置空表 setnull(v)：将线性表 v 置为一个空表。

② 求线性表的长度 length(v)：求得线性表 v 的长度。

③ 求线性表的第 i 个数据元素 get(v, i, *x)：取得线性表 v 中第 i 个数据元素 a_i（$0 \leq i \leq$ length(v)-1），x 为返回的数据元素。

④ 取某数据元素的直接前驱 prior(v, i, *x)：取得线性表 v 中数据元素 a_i 的直接前驱元素 a_{i-1}（$1 \leq i \leq$ length(v)-1），x 为返回的数据元素。

⑤ 取某数据元素的直接后继 next(v, i, *x)：取得线性表 v 中数据元素 a_i 的直接后继元素 a_{i+1}（$0 \leq i \leq$ length(v)-2），x 为返回的数据元素。

⑥ 在线性数据表中查找某个特定的值 locate(v, x)：返回线性表 v 中元素值等于 x 的第一个数据元素 a_i 的序号 i，若线性表 v 中不存在值为 x 的数据元素，则函数值为-1。

⑦ 在线性表 v 中第 i 个位置插入一个新的数据元素 insert(v, i, x)($0 \le i \le n$)：在线性表 v 的第 i-1 个数据元素和第 i 个数据元素之间（即在第 i 个数据元素之前）插入一个新的数据元素 x，得到一个长为 n+1 的新线性表

$$(a_0, a_1, \cdots, a_{i-1}, x, a_i, \cdots, a_{n-1}) \tag{2.2}$$

⑧ 删除线性表 v 中第 i 个数据元素 delete(v, i)：删除线性表 v 中第 i 个数据元素 a_i，得到一个长度为 n-1 的新线性表

$$(a_0, a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_{n-1}) \tag{2.3}$$

式中，$0 \le i \le n$-1。

利用以上八种基本运算，可以实现其他更复杂的运算，例如线性表的合并、拆分、复制等。

2.2　线性表的存储结构

上一节我们学习了线性表中数据元素之间的逻辑结构，那么线性表这种数据结构在计算机内怎样表示呢？这就涉及了数据的物理存储结构，数据的物理存储结构有顺序存储结构和链式存储结构两种。下面分别讨论线性表的这两种存储结构。

2.2.1　线性表的顺序存储结构

我们把线性表（$a_0, a_1, \cdots, a_{n-1}$）中的数据元素按照它在线性表中出现的先后顺序依次放入一组地址连续的存储单元中，即用数据元素在计算机内物理位置上的相邻关系来表示线性表中数据元素之间相邻的逻辑关系。我们把线性表的这种机内表示称为线性表的顺序存储结构，如图 2.1 所示。

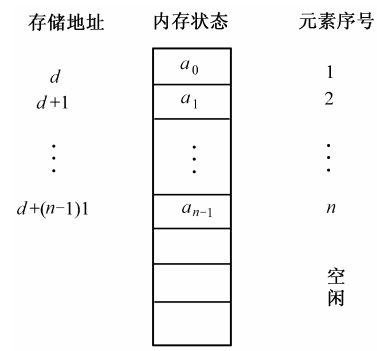


图 2.1　线性表的顺序存储结构

显然，线性表中逻辑上相邻的数据元素在内存中也处于相邻的物理位置。因此，我们只要知道线性表中第一个数据元素的存储地址，就能计算出其他数据元素所在的物理位置。假设线性表的每个元素需占用 l 个存储单元。第一个元素所在位置为 d_0，则线性表中下标为 i 的数据元素（第 i+1 个数据元素）的存储位置为 d_i=d_0+i。这样我们就可以任意存放和读取线性表中的一个数据元素。

我们知道在计算机程序语言中，所有的数组在计算机内均由连续的存储单元存储，最低地址对应于数组的第一个元素，最高地址对应于数组的最后一个元素。这个存储特点正好符合线性表顺序存储结构的存储要求，因此我们可以用数组来描述线性表的顺序存储结构。

```
#define MAX_LENGTH 100              // 线性表可能的最大长度值
typedef struct _sequenList{
    int elements[MAX_LENGTH];       // 存放线性表中的数据元素
```