普通高等教育 “十三五”创新型规划教材
理论+实践+数字资源一体化

主编 马宏茹 吴璞 姚保峰

数据结构

<<<<< SHU JU JIE GOU

中国矿业大学出版社
China University of Mining and Technology Press
国家一级出版社
全国百佳图书出版单位

数据结构

主　编　马宏茹　吴　璞　姚保峰
副主编　殷晓玲　周　昊

中国矿业大学出版社

图书在版编目（CIP）数据

数据结构 / 马宏茹，吴璞，姚保峰主编.
—徐州：中国矿业大学出版社，2018.8
ISBN 978-7-5646-4045-3

Ⅰ.①数… Ⅱ.①马… ②吴… ③姚… Ⅲ.①数据结构
—教材 Ⅳ.①TP311.12

中国版本图书馆 CIP 数据核字（2018）第 153066 号

书　　名　数据结构
主　　编　马宏茹　吴　璞　姚保峰
责任编辑　陈　慧
出版发行　中国矿业大学出版社有限责任公司
　　江苏省徐州市解放南路　邮编 221008
营销热线　（0516）83885307　83884995
出版服务　（0516）83885767　83884920
网　　址　http：//www.cumtp.com　**E-mail**：cumtpvip@cumtp.com
印　　刷　武钢实业印刷总厂
开　　本　787×1092　1/16　**印张** 18　**字数** 442 千字
版次印次　2018 年 8 月第 1 版　2018 年 8 月第 1 次印刷
定　　价　48.00 元
（图书出现印装质量问题，本社负责调换）

前 言

“数据结构”是计算机及相关专业的专业基础课和核心课程。随着计算机应用范围逐渐深入各个学科领域，整个社会对具有计算机专业技能和实用技术的应用型人才的需求更加迫切。“数据结构”所研究的知识内容和技术方法，无论对学习计算机学科的其他相关课程，还是对从事软件设计和开发工作，都是重要的理论基础。

本书的设计采用了“项目导入、项目启发”的思想。项目导入、项目启发的思路与教育部推导的“作中学，作中教”的“工学结合”思想相吻合，从“数据结构”课程的特点来讲，特别适合于按照“工学结合”的思想来构建其知识架构。本书在推进工学结合人才培养模式改革方面具有重要意义，可以作为普通高等院校计算机及相关专业的教材，也可以作为软件开发者的参考资料。

在内容选取上，本书符合复合型、应用型人才培养目标的要求，遵循教学规律和认知规律。每个章节都由项目导入开篇，引导读者直观了解学习本章节内容能解决的实际问题。把抽象问题具象化，易于理解，循序渐进地引导读者理解和掌握核心知识。每章导入的案例都是经过精心设计，选取具有代表性和典型性的实例。在知识讲解中，本书力求做到化繁为简，复杂问题简单化，由浅入深地剖析。项目结合更是本书的特色，从程序开发的思想分析本项目，一步步启发读者设计完成经典问题，进而达到灵活应用的目标。实作强化部分列举本章节经典问题并给出解决方法，培养思维的全面性。最后通过精选练习达到强化巩固的目标。

本书案例安排循序渐进，通俗易懂，并有习题辅助，实用性强。全书共包括 8 章：第 1 章介绍数据结构和算法的基本概念；第 2 章讲解线性表的逻辑结构、存储结构、线性表的基本操作及应用；第 3 章讲解栈和队列的基本概念、逻辑结构、存储方式、基本操作及应用；第 4 章讲解串和数组的存储结构、基本算法及应用；第 5 章讲解树的基本概念、二叉树的存储结构及其基本操作及应用；第 6 章讲解图的基本概念、图的存储结构、遍历及应用；第 7 章讲解排序的基本概念，并介绍常用的排序算法；第 8 章讲解查找的基本概念，并介绍常用的查找算法。

本书的每位编者都有丰富的计算机教学经验和项目开发实战经验。其中第2章、第3章、第7章由马宏茹编写；第8章由吴璞编写；第1和第4章由姚保峰编写；第5章由殷晓玲编写；第6章由周昊编写。全书由马宏茹负责统稿。

由于编者水平有限，再加之时间仓促，本书仍存在错误及不妥之处，恳请广大读者批评指正。

编　者

2018年5月

目 录

第1章 数据结构与算法

学习目标

- 理解数据结构的基本概念；
- 理解数据的逻辑结构和物理结构；
- 理解算法的概念；
- 理解算法和数据结构的关系；
- 掌握算法的时间复杂度和空间复杂度的分析方法。

1.1 项目导入

随着计算机产业的飞速发展，电子计算机技术已经逐步融入人们的生活和工作中，计算机的主要任务也从早期的数值计算转变为更多的完成非数值计算任务。现在，不管是生活中的购物、出行还是工作中的数据处理、工业控制和信息管理都可以由计算机帮助我们完成。计算机在迅速改变人们活动方式的同时，也对处理问题的方法提出了新的要求。在完善的人工智能到来之前，计算机本身只是一台拥有超高运算速度和存储容量的机器，没有人的指挥，它是不能做任何事情的，而要想指挥计算机完成各种各样的任务，就要为计算机编写好特定的“程序”。为了让计算机能把任务做好，就必须写出“好”的程序，这就要求人们在处理问题之前，认真地分析任务中有哪些数据需要处理，以及这些数据之间有什么样的关系，然后把这些数据以最适当的方式表示出来。要做到这一点，就必须学习“数据结构”。请看如下案例。

【例1.1】学生学籍信息管理系统。

每年高校有新生入校时，学校都会把学生的信息录入到教务处的学籍管理系统中，以便对学生信息进行统一管理。在学籍管理系统中显示的学生信息如表1-1所示。

表 1-1　学生学籍信息表

学号	姓名	专业	性别	联系电话	籍贯
201710001	张琳	计算机科学与技术	男	13812345678	北京
201710002	李雷	计算机科学与技术	男	15612345678	上海
201710003	韩梅梅	计算机科学与技术	女	15812345678	天津
201710004	赵菲菲	计算机科学与技术	女	17012345678	重庆

对于以上的信息，计算机应当如何表示和处理呢？可以看出，所有学生的信息都按照学号顺序依次排列，构成了学生学籍信息的线性序列，显然各个学生的信息之间是一种“一对一”的线性关系，数据之间的这种结构是一种“线性”的数据结构。

【例 1.2】学校的组织结构。

任何企事业单位都会有自己的一套管理架构，目前大多数高校为了便于工作开展，都会分设若干教学部门、教辅机构和职能部门，如图 1-1 所示。

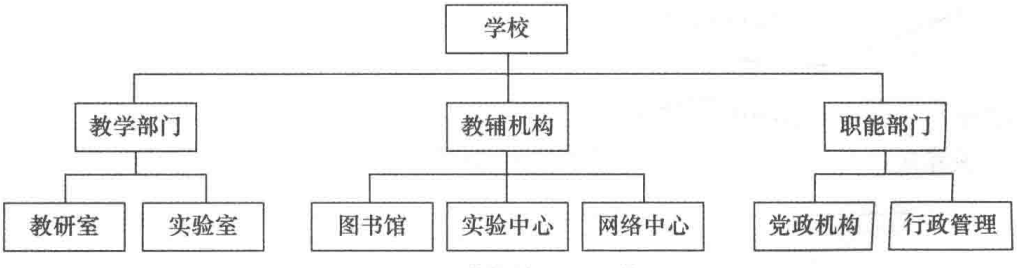


图 1-1　学校的组织架构图

事实上，不管是企事业单位的组织结构管理，还是计算机中的文件系统管理，都具有类似的特征，就是它们所表示的内容之间具有明确的层次关系，这种关系显然不是线性的。在图 1-1 中，从最顶层的位置(如学校)出发，每一个位置(如教学部门)都有若干个分支，直到最底层(如实验室)不再有分支，这种结构非常像“一棵倒立的树”，学校相当于树根的位置，教学部门相当于树枝，而实验室则是树叶。在这种结构中，各位置的数据之间存在着“一对多”的层次关系，数据之间的这种结构就是一种“树”形的数据结构。

【例 1.3】最短路径选择问题。

外卖小哥要从 A 地出发送餐到 C 地，如图 1-2 所示，有多条路径可以选择？路径上标识的数值是两地之间的到达距离，请问走哪条路能最快到达目的地？这类问题中的数据之间关系显然更加复杂，因为任何两点间都可能有路径相连，因此该问题的可选路径有很多。在这种结构中，各位置的数据之间存在着“多对多”的对应关系，数据之间的这种结构就是一种“图”状的数据结构。

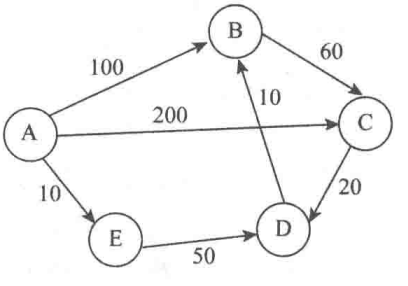


图 1-2　最短路径选择

从以上三个例子可以看出，在我们用计算机处理问题时，问题中所涉及的数据是多种多样的，不同问题中数据之间的关系也各不相同，在解决

各种问题时，就必须认真分析我们所面对的问题涉及哪些数据，以及这些数据之间存在哪一种关系，只有这样，我们才能正确地表示数据并合理地给出解决问题的方案。事实上，现实世界中的问题虽然多种多样，但是问题中所涉及的数据和关系一般都可以归纳为以上的三种结构，即线性结构、树结构和图结构。因此，我们如果能够归纳出以上几种结构的特点及它们的表示和处理方法，那么我们面对任何问题时就有了基本的想法，而这就是数据结构所研究的问题。因此，数据结构是一门研究非数值问题中所处理的数据、数据之间的关系以及相关操作实现方法的课程。

1.2　知识概述

1.2.1　数据结构的基本概念

为了方便后续的讲解，我们首先介绍数据结构的一些相关术语。

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能输入计算机中并被计算机程序处理的符号的总称。随着计算机的快速发展和应用，数据所涵盖的概念也越来越广，可以说包括数值、字符、声音、图形图像、动画视频等一切能够被计算机接受的符号集合都属于数据的范畴。数据是计算机程序加工的“原料”。

数据元素(Data Element)是组成数据的基本单位，是数据集合的个体，在计算机中通常作为一个整体进行考虑和处理。如表 1 - 1 中的每一行数据都是一个数据元素，表示一位学生的基本信息；图 1 - 1 中的每个部门(如实验室)和图 1 - 2 中的每个位置(如 A)都是一个数据元素。数据元素有时也称为元素、结点或记录。

数据项(Data Item)是组成数据元素的、有独立含义的最小单位。一个数据元素由若干个数据项构成，如表 1 - 1 中每个学生的学号、姓名、性别都是数据项。

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，整数数据对象是集合 N = {0，±1，±2，…}，字母字符数据对象是集合 C = {‘A’，‘B’，‘C’，…，‘Z’}。表 1 - 1 中的学生的学籍信息表也可看作一个数据对象。可以看出，不论数据元素是无限集(如整数集)、有限集(如字符集)，还是由多个数据项组成的复合数据元素(如学籍表)，只要性质相同，都属于同一个数据对象。

数据结构(Data Structure)是指相互之间存在一种或多种特定关系的数据元素的集合。可以理解为，数据结构中的“数据”是指数据元素的集合，而“结构”是指数据元素间关系的集合。因此，计算机所处理的数据不仅包含数据集合本身，更包含集合中数据之间的内在联系。数据结构的形式通常定义为一个二元组：

Data_Structure = (D，R)

其中，D 是数据元素的集合，R 是 D 上关系的有限集。

图 1 - 3 所示是一个用数组存储的线性表，其数据结构可以定义为：

d1	d2	d3	d4	d5

图 1 - 3　数组存储的线性表

List = (D, S)

D = {d1, d2, d3, d4, d5}　　//数据元素的集合

S = { < d1, d2 > , < d2, d3 > , < d3, d4 > , < d4, d5 > }　　//数据元素之间的关系

这里 < x，y > 的形式表示“x 领先于 y”的关系，即 x 是 y 的前驱结点，y 是 x 的后继结点，因此 < d1，d2 > 表示 d1 和 d2 的关系是 d1 在 d2 之前，即 d1 是 d2 的前驱结点，d2 是 d1 的后继结点。

1.2.2　数据的逻辑结构与存储结构

1. 数据的逻辑结构

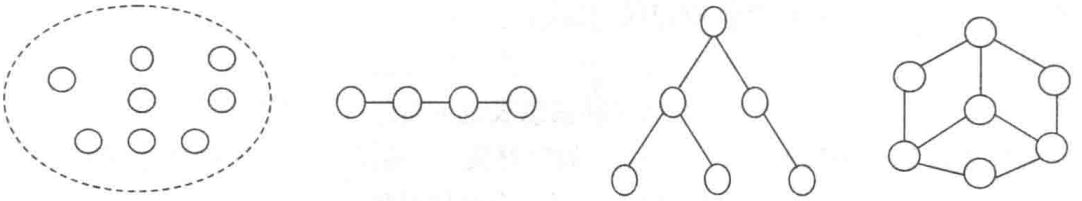
根据数据元素之间关系的不同特性，通常数据元素之间的关系可以表示为四种结构，分别是集合结构、线性结构、树形结构和图状结构，数据元素之间的关系也就是数据的逻辑结构。数据的逻辑结构(Logical Structure)是从逻辑关系上描述数据，它与数据的存储无关，体现了数据元素之间的抽象化关系。因此，数据的逻辑结构可以看作是从具体问题抽象出来的数学模型，它是独立于计算机的。

① 集合结构。数据元素之间除了“属于同一个集合”的关系外，别无其他关系。如图 1 -4(a)所示。

② 线性结构。数据元素之间存在一对一的关系。例如，学生学籍信息表中的数据是按学号顺序排列的，就组成了一个线性结构。如图 1 -4(b)所示。

③ 树形结构。数据元素之间存在一对多的关系。例如，在学校的组织架构图中，学校包含多个部门，每个部门又包含多个子部门，就构成了树形结构。如图 1 -4(c)所示。

④ 图状结构。数据元素之间存在多对多的关系。例如，多个位置之间的路径选择问题，任何两个位置都可能是连通的，从而构成了图状结构。图状结构也称为网状结构。如图 1 -4(d)所示。



(a) 集合结构　(b) 线性结构　(c) 树形结构　(d) 图状结构

图 1 -4　四类基本的逻辑结构

在以上四种结构中，集合结构中的任何两个数据元素之间都没有逻辑关系，组织形式极为松散，因此通常用其他结构来表示。因此，数据的逻辑结构分为两大类，即线性结构和非线性结构，线性结构包括线性表、栈、队列、串、数组和广义表，非线性结构包括树形结构和图状结构，如图 1 -5 所示。

2. 数据的存储结构

计算机在处理数据时，必须先将数据存放在内存中进行存储。对于不同逻辑结构的数据，它们在计算机中的存储方式也有所不同。数据的存储结构(Storage Structure)是数据在

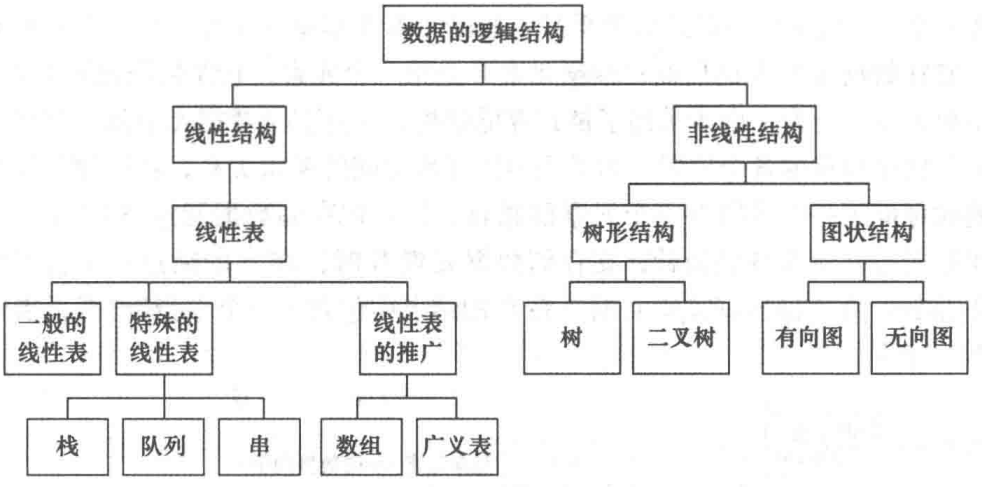


图 1－5　数据的逻辑结构分类

计算机中的表示，是逻辑结构在计算机中的实现。数据的存储结构包括数据元素的表示和关系的表示。数据的存储结构也称为数据的物理结构。

数据在内存储器中通常有两种存储方式，一种是把数据集中存放在内存中的一片连续的存储空间；另一种是将数据分散地存储在内存的各个位置。前者称为顺序存储结构，后者称为链式存储结构。

（1）顺序存储结构

顺序存储结构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。采用顺序存储结构时，数据在内存中被分配一片连续的存储空间，按照数据之间的逻辑关系顺序存放每个数据。

数组就是一种常用的顺序存储结构，若定义一个数组：int a[10]，设每个元素占 4 个字节，则从 a[0]到 a[9]的每个元素在内存中的存储情况如图 1－6 所示。可以看出，数组中的每个元素在内存中是按顺序连续存放的，数组元素的存储位置本身就体现了它们之间的逻辑关系。因此，使用这种存储结构，数据只需存储数据元素本身，无须包含反映数据之间关系的具体信息，可以说顺序存储结构对存储空间的利用率是 100%，即所有分配给数据元素使用的存储空间全都被用来存放数据内容。

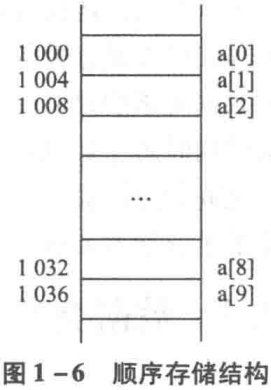


图 1－6　顺序存储结构

（2）链式存储结构

链式存储结构借助每个元素的指针来表示数据元素之间的逻辑关系。每个数据元素在存储时均由两个部分组成，一部分用来存放数据元素自身的值，另一部分用来存放与当前元素相邻的数据元素的位置信息，即每个数据元素包含数据部分和指针部分。采用链式存储结构时，无须为数据元素分配连续的存储空间，每个数据元素可以被存储在内存中的任何可用位置，只需通过指针来表示元素之间的逻辑关系。

图 1－7(a)所示为一个链式存储结构的线性表，表中每个数据元素的结构如图1－7(b)所示，均包括数据部分(data)和指针部分(next)。图 1－7(a)中包含三个数据元素 A、B

和 C，这三个数据元素之间的逻辑关系是：数据元素 A 是第一个元素；数据元素 B 是第二个元素，它在数据元素 A 的后面；数据元素 C 是第三个元素，它在数据元素 B 的后面，C 的后面不再有其他元素。由于采用了链式存储结构，因此三个数据元素没有连续存储，而是分散地存储在内存的各个位置。为了表示出元素之间的逻辑关系，在存储每个数据元素时，都同时存储了一个指向下一个元素的指针，即：在存储数据元素 A 时，在 A 的结点中包含了指向数据元素 B 的指针；在存储数据元素 B 时，在 B 的结点中包含了指向数据元素 C 的指针；在存储数据元素 C 时，在 C 的结点中包含了一个空指针“^”，表示 C 是最后一个数据元素。

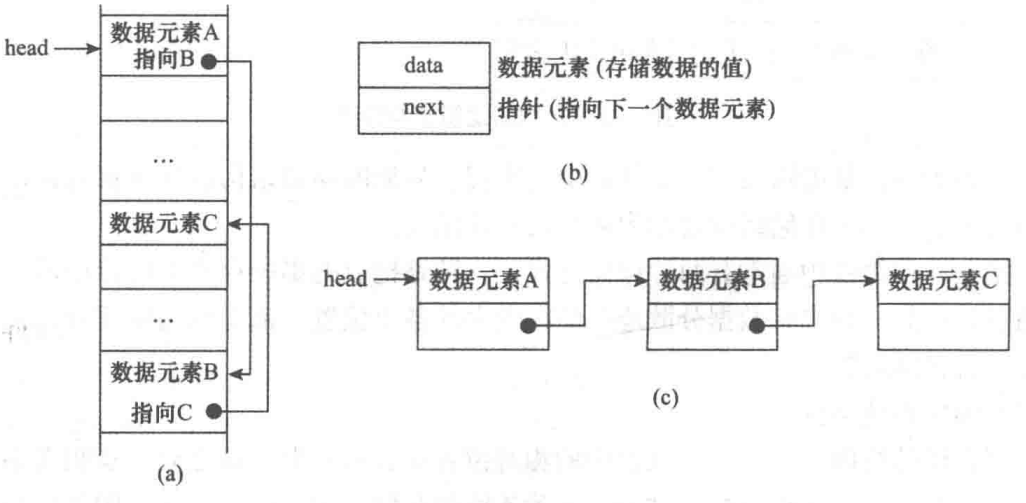


图 1-7　链式存储结构

可以看出，链式存储结构是通过指针来体现数据元素之间的逻辑关系的，所以数据元素的存储可以是分散的，不需要像顺序存储那样动用大的连续存储空间，因此采用链式存储时数据元素的存储就更为灵活，可以有效地提高对内存空间的利用率。但是，采用链式存储结构时每个数据元素除了要存储数据本身，还要另外开辟存储空间来存储指针以反映元素之间的逻辑关系，这也构成了存储的额外开销，因此从这个意义上说，链式存储结构又降低了存储空间的利用率。

1.2.3　数据类型

1. 数据类型

数据类型(Data Type)是一组性质相同的值集合以及定义在这个值集合上的一组操作的总称。在程序设计语言中，每个数据都属于某种数据类型，数据类型显示或者隐式地规定了数据的取值范围、存储方式以及允许进行的运算。例如 C 语言中的整型变量，其值集为整数集合，定义在其上的操作包括加、减、乘、除和求余等算术运算。因此，程序设计语言中的数据类型实际上就是已经实现的数据类型的实例。

按“值”的不同特性，程序设计语言中的数据类型可以分为原子类型和结构类型两类。原子类型的值是不可再分的，如 C 语言中的整型、实型和字符型等；结构类型的值由若干成分按某种结构组合而成，因此是可以分解的，它的成分可以是原子的，也可以是结构

的，如 C 语言中的结构体类型。

数据类型和数据结构的概念是密切相关的，如顺序存储结构可以借助程序设计语言的数组类型加以描述，链式存储结构可以借助指针类型加以描述。

2. 抽象数据类型

抽象就是抽取出问题的本质特征，而忽略非本质的细节。在计算机内使用的是二进制数，汇编语言中则可以给出各种数据的十进制表示，它们是二进制的抽象，程序设计人员可以在编程时直接使用，而不必考虑实现细节。在高级语言中，则给出更高一级的数据抽象，出现了数据类型，如整型、实型、字符型等，可以进一步利用这些类型构造出线性表、栈、队列、树、图等复杂的抽象数据类型。

抽象数据类型(Abstract Data Type，ADT)定义了一个数据对象、数据对象中各元素的结构关系以及一组处理数据的操作。抽象数据类型和数据类型实质上是一个概念。例如，各个计算机都拥有的“整数”类型是一个抽象数据类型，尽管它们在不同处理器上实现的方法可以不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此“抽象”的意义在于数据类型的数学抽象特性。

相对于数据类型，抽象数据类型的范畴更广，它不再局限于前述各处理器中已定义并实现的数据类型，还包括用户在编写程序时自己定义的数据类型。为了提高软件的复用率，近代的程序设计方法学要求一个程序的框架应建立在数据之上，而不是建立在操作之上，即在编写程序的每个相对独立的模块上，定义一组数据和施加于这些数据上的一组操作，并在模块内部给出这些数据的表示及其操作的细节，而在模块外部使用的只是抽象的数据和抽象的操作。显然，所定义的数据类型的抽象层次越高，含有该抽象数据类型的软件模块的复用程度也就越高。一个含抽象数据类型的程序模块通常应包含定义、表示和实现三个部分。

抽象数据类型的定义格式为：

ADT 抽象数据类型名{

　　数据对象：<数据对象的定义>

　　数据关系：<数据关系的定义>

　　基本操作：<基本操作的定义>

}ADT 抽象数据类型名

其中数据对象和数据关系的定义采用数学符号和自然语言描述，基本操作的定义格式为：

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>

基本操作有两种参数，赋值参数只为操作提供输入值；引用参数以“&”打头，除可提供输入值外，还将返回操作结果。“初始条件”描述了操作执行之前数据结构和参数应满足的条件，若初始条件为空，则省略。“操作结果”说明了操作正常完成后，数据结构的变化状况和返回的结果。

【例 1.4】一元二次多项式 ax^2+bx+c 的抽象数据类型定义。

假定起名为 Quadratic，该类型的数据部分为三个系数项 a，b 和 c，操作部分为：

① 初始化 a，b 和 c 的值，假定它们默认值均为 0；

② 做两个多项式的加法，返回它们的和；

③ 根据给定 x 的值计算多项式的值，并返回；

④ 计算机求方程：$ax^2+bx+c=0$ 的根，对于有实根、无根和不是二次方程的情况要分别返回不同的值；

⑤ 按照 ax^2+bx+c 的格式输出二次多项式，在输出时要注意去掉系数为 0 的项，并且当 b 和 c 的值为负时，其前不能出现加号。

```
ADT  Quadratic{
数据对象：D={a，b，c | a，b，c∈RealSet，a≠0}
数据关系：R1={a 是二次项的系数，b 是一次项的系数，c 是零次项系数}
基本操作：
InitQuadratic(&Q，x，y，z)                    // 初始化一元二次多项式
操作结果：构造一个一元二次多项式 Q，x，y，z 的值分别被赋值给参数 a，b，c。
DestroyQuadratic(&Q)
操作结果：销毁一个一元二次多项式 Q。
Add(&QA，Q1，Q2)                              // 两个多项式的加法
初始条件：一元二次多项式 Q1 和 Q2 已存在。
操作结果：求两个一元二次多项式的和放入 QA 中。
Eval(Q，x，&e)                                // 一元二次多项式求值
初始条件：一元二次多项式 Q 已存在。
操作结果：代入 x 求一元二次多项式的值并将结果存入 e 中。
Root(Q，&r1，&r2)                             // 求一元二次项式的根，两个实根
                                                由引用参数 r1，r2 带回
初始条件：一元二次多项式 Q 已存在。
操作结果：求一元二次多项式的根。
Print(Q)                                      // 输出二项式
初始条件：一元二次多项式 Q 已存在。
操作结果：输出一元二次多项式，在输出时要注意去掉系数为 0 的项，并且当 b
和 c 的值为负时，前面不能出现加号。
}
```

1.2.4 算法和算法分析

瑞士著名的计算机科学家、图灵奖获得者、Pascal 语言设计者 N. Wirth 教授给出了一个著名的公式：

$$\text{算法} + \text{数据结构} = \text{程序}$$

可以看出，数据结构和算法是程序设计的两大要素，二者相辅相成、缺一不可。

1. 算法的定义和特性

算法(Algorithm)是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令。一个算法必须满足以下五个方面的特性：

① 有穷性。算法的有穷性是指算法必须能在执行有限个步骤之后终止。

② 确定性。算法的每一步骤必须有确切的定义。

③ 输入。一个算法有0个或多个输入，以刻画运算对象的初始情况。所谓0个输入是指算法本身给出了初始条件。

④ 输出。一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

⑤ 可执行性。算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步的，即每个计算步都可以在有限时间内完成(也称之为有效性)。

2. 算法设计的要求

一个好的算法应该符合以下几个要求：

① 正确性。算法的正确性是指算法至少应该具有输入、输出和加工处理无歧义性、能正确反映问题的需求、能够得到问题的正确答案。大体分为以下四个层次：

a. 算法程序没有语法错误。

b. 算法程序对于合法输入能够产生满足要求的输出。

c. 算法程序对于非法输入能够产生满足规格的说明。

d. 算法程序对于故意刁难的测试输入都有满足要求的输出结果。

② 可读性。设计算法的目的，一方面是为了让计算机执行，但还有一个重要的目的就是为了便于他人的阅读，让人理解和交流，自己将来也可阅读。如果可读性不好，时间长了自己都难以理解，所以可读性是评判算法(也包括实现它的程序代码)好坏很重要的标志。可读性不好不仅无助于人们理解算法，晦涩难懂的算法往往隐含错误，不易被发现并且难以调试和修改。

③ 健壮性。当输入的数据非法时，算法应当恰当地做出反应或进行相应处理，而不是莫名其妙地输出结果。并且处理出错的方法不应是中断程序的执行，而应是返回一个表示错误或错误性质的值，以便于在更高的抽象层次上进行处理。

④ 高效率与低存储量。通常，算法的效率指的是算法的执行时间；算法的存储量指的是算法执行过程中所需要的最大存储空间，两者的复杂度都与问题的规模有关。

3. 算法分析

一个问题往往可以采用多种不同的算法进行求解，前面已经分析了一个好的算法应该符合的四点要求，在以上列出的四点要求中，可读性和健壮性难以进行量化的评价，因此，对于解决一个问题的不同算法，对其优劣的评价主要是在保证算法正确性的基础上，考察算法的效率和存储占用情况。在数据结构中，对一个算法执行效率的度量，称为时间复杂度；对一个算法在执行过程中所需占用存储空间的度量，称为空间复杂度。

(1) 算法的时间复杂度

对算法执行时间的度量通常有事后统计法和事前分析估算法两种方法。

① 事后统计法。这种方法要求先设计解决问题的算法，并根据算法实现程序，然后使用程序中的计时功能，精确计算使用不同算法在输入同样的测试数据时程序的实际执行时间。这种方法存在三个方面的问题：一是必须先根据算法编写程序，对于复杂问题而言代价较大；二是所计算的时间受计算机的硬件、软件等环境因素影响，得到的结果不够准确；三是选择的测试数据不同，测试结果可能有较大差异，从而难以对算法的优劣进行准确判断。因此，在对算法的优劣进行度量时，通常不采用该方法。

② 事前分析估算法。程序在计算机上执行时所消耗的时间主要取决于以下几个因素：

a. 采用的算法策略。

b. 问题的规模，即要处理的数据量。

c. 机器执行指令的速度。

d. 编译程序产生的机器代码的质量。

在以上几个因素中，后两者主要取决于计算机的硬件条件和软件环境，因此抛开这些外部的相关因素，可以认为一个算法的时间效率将只依赖于问题的规模，也就是说，在不考虑外部因素影响的情况下，算法的时间效率取决于问题的规模。事实上，程序的实际执行时间取决于指令的执行次数，因此可以用一个指令执行次数(可以用算法基本操作的执行次数表示)与问题的规模的函数来表示算法的时间效率。

例如，求 1 ~ 100 的和问题，可以有以下两种不同的算法。

第一种算法采用循环求解：

```
sum = 0; n = 100;                    //执行 1 次
for (i = 1; i <= n; i + +)           //执行 n + 1 次
{
    sum + = i;                       //执行 n 次
}
```

第二种算法采用数学家高斯提出的方法求解：

```
sum = 0;                             //执行 1 次
n = 100;                             //执行 1 次
sum = (1 + n) *n/2;                  //执行 1 次
```

对于以上两种算法，可以粗略地计算两种方法的语句执行次数以代替实际的指令执行次数，从而比较两种算法的性能。第一种方法共执行了 2n + 2 次，而第二种方法只执行了 3 次。第一种方法的执行次数随着问题输入规模的增加而增加，而第二种方法的执行次数与 n 无关。显然，第二种算法的时间效率更高。

从上面的例子可以看出，计算算法时间的最可靠方法就是计算对运行时间有影响的语句执行次数，由此得到一个语句执行次数和问题的规模的函数。通过该函数就可以看出，随着问题输入规模 n 的增大，语句执行次数的差异决定了算法的时间效率。

【例 1.5】求两个 n 阶方阵的乘积 C = A × B。

```
#define N 100
void matrix(int a[][], int b[][], int c[][])
```