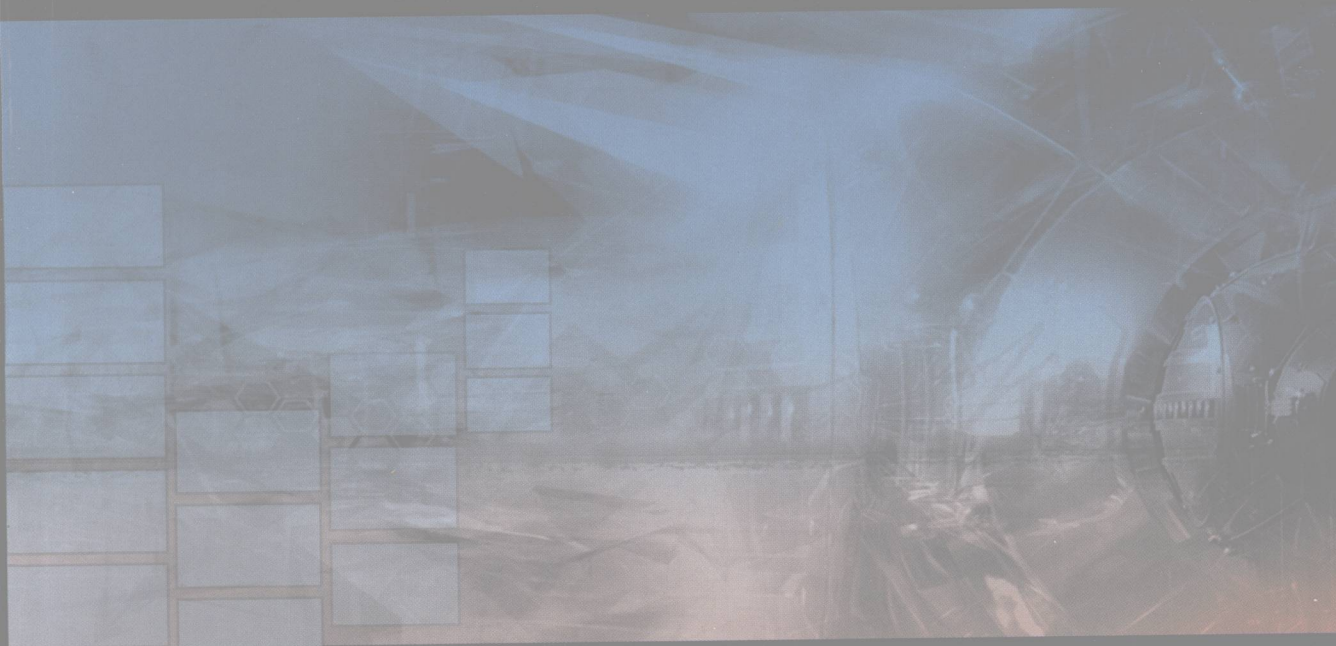
普通高等教育“十一五”国家级规划教材

高等学校计算机系列



数据结构（C++版）

杨秀金　主编

人民邮电出版社
POSTS & TELECOM PRESS

普通高等教育“十一五”国家级规划教材

高等学校计算机系列

数据结构（C++版）

杨秀金　主编

人民邮电出版社

北京

图书在版编目（CIP）数据

数据结构：C++版 / 杨秀金主编. —北京：人民邮电出版社，2009.4
普通高等教育“十一五”国家级规划教材. 高等学校计算机系列
ISBN 978-7-115-19534-0

Ⅰ. 数… Ⅱ. 杨… 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. TP311.12 TP312

中国版本图书馆CIP数据核字（2009）第022439号

内 容 提 要

本书根据教育部高等学校计算机科学与技术教学指导委员会关于“数据结构”课程的指导性大纲进行编写。书中系统地介绍各种数据结构的特点、存储结构及相关算法，并采用面向对象C++语言描述数据结构和算法。主要包括：数据结构的基本概念、算法描述和算法分析初步，线性表、栈、队列、串、数组、树、图等数据结构，以及排序、查找等内容。多数章节给出了完整C++语言源程序示例，每章后面配有小结和习题。最后一章介绍怎样编写数据结构的应用程序及实验步骤规范。

本书叙述清晰、深入浅出、注重实践和应用，便于教学。

本书可作为普通高等学校计算机及相关专业本科或专升本的教材，也可供相关证书考试、考研或从事计算机应用与工程工作的科技工作者自学参考。

普通高等教育“十一五”国家级规划教材
高等学校计算机系列
数据结构（C++版）

◆ 主　　编　杨秀金
责任编辑　邹文波
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京世纪雨田印刷有限公司印刷
◆ 开本：787×1092　1/16
印张：20
字数：490千字　　2009年4月第1版
印数：1－3 000册　　2009年4月北京第1次印刷

ISBN 978-7-115-19534-0/TP

定价：32.00元

读者服务热线：(010)67170985　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

“数据结构”是计算机专业重要的专业基础课程与核心课程之一。

为适应我国计算机科学技术的应用和发展，进一步提高计算机专业“数据结构”课程的教学质量，作者根据多年的教学经验，结合当前高等教育大众化的趋势，在分析国内、外多种同类教材的基础上，编写了本书。

在2000年，作者曾经编写出版了《数据结构》（使用C语言）一书，在2004年该书的第2版出版发行，前后印刷十多次。随着计算机应用的发展，作者又编写了采用C++语言的《数据结构——使用C++语言》一书。

本书继承了前几本书的一些特色，作者结合近几年教学改革的实践，对其内容做了进一步的优化、补充和完善。近几年教学实践表明，在教学中较早引入面向对象概念和技术，对本科的工程应用型人才培养是有益的。实践还表明，采用面向对象的C++语言进行数据结构教学是可行的，且对于普通高校本科学生也是适用的。

本书具有以下特色。

1．实例导入，由浅入深。在概念引入时，尽量采用人们熟悉的计算机应用实例，使读者感到“有用”。对数据结构概念和理论的介绍，适当增加了图示，使复杂抽象的概念尽量形象化，使读者感到“可学”。对各种基本算法描述尽量详细，叙述清楚。对数据结构的基本概念、基本理论的阐述注重科学严谨。为了逐步提高学生的抽象思维能力，与作者先期几本书相比，在本书中增加了线性表等典型数据结构的ADT描述。

2．理论联系实际。在每章中，配合核心理论的阐述，一般都配有实例和例题。与先前的几本书相比，某些内容有所补充。例如，算法时间复杂度 $T(n)=O(f(n))$ 的分析，是重点也是难点。在第1章给出了几种典型时间复杂度归纳计算的具体示例，在对影响算法效率主要因素的提取和对次要因素忽略方面，描述比较详细。再如，对查找表的查找成功（或不成功）平均查找长度的计算，在第8章中给出了具体示范和分析。

为了巩固所学的理论知识，每章都附有习题，供学生书面练习和上机作业选用。针对部分学生中存在的“虽懂概念，但不会编程”问题，每章在介绍完某种数据结构的类设计后，一般给出几条程序语句，以示范对类对象的定义和对算法的调用，并且在章的后面还有一节完整的C++语言源程序示例，供读者参考模拟，以便提高程序设计能力。

“数据结构”课程的一个重要任务是培养学生进行复杂程序设计的能力，因此，在第11章数据结构程序设计中，给出了从“问题”到“程序”的一般过程，同时给出了上机实验步骤规范，并配有两个上机实验示范例题，目的在于培养学生进行规范化程序设计的素养，提高学生结合实际解决问题的应用能力。

3．采用C++语言实现数据结构。当前“数据结构”课程算法描述的语言工具并不统一，主要有C、C++和Java 3种。由于学生到高年级必须学习和运用面向对象技术，与其先使用面向过程的C语言描述算法，不如直接采用面向对象的C++语言。采用面向对象技术进行软件设计已是大势所趋，采用C++语言描述算法和进行程序设计有利于学生工程应用能力的培

养。每种典型的数据结构都有自己的抽象数据类型 ADT 定义，而 C++类的设计正好与 ADT 的编程实现相吻合。使用类的成员函数来表示算法，使得函数接口的形参表（比 C 语言）更加简洁。虽然书中数据结构的算法描述和程序设计大多采用面向对象的方法，但本书并不是为彻底解决面向对象的问题而编写的。

4. 注重基础、循序渐进。由于采用了 C++语言描述数据结构，对于低年级学生的学习存在一定的难度。为了使读者更好地学习数据结构自身的知识内容，减轻语言工具所带来的困难，本书对相关各章节的内容作了独特处理。

第 1 章绪论，在 1.4 节简要介绍了学习“数据结构”课程所必须的 C++语言的基本知识。

第 2 章线性表，这是后续章节的重要基础，因此对两种典型存储结构（顺序存储结构、链表存储结构）的介绍显得十分重要。在 2.2.1 小节，首先在不考虑面向对象的条件下，对顺序存储结构（一维数组）的基本概念进行详细介绍，然后再将该结构放在面向对象的类中加以讨论。在 2.3.1 小节，首先在不考虑面向对象的条件下，对链表的基本概念和指针的基本操作以图文结合的方式进行详尽介绍，然后转入链表类的定义和面向对象的程序设计，便于学生理解掌握。

第 3 章栈和队列，注重介绍栈和队列自身的基本概念，并且注重巩固 C++类的基本概念，暂不涉及类模板等复杂概念。为了应对第 6、7 章不可避免地使用栈和队列类模板的需要，本书在第 10 章专门对栈和队列类的模板及其运用进行了详细介绍，供读者自学参考。

5. 难点处理，避繁就简。C++的函数重载、函数模板在数据结构的学习中是不可避免的。为了不让 C++语法难点影响数据结构基本概念的学习，在基础章节中尽量不用或少用它们，在非用不可的章节，则以“够用”为尺度。

对于函数重载，首先在 1.4.1 小节对此概念进行初步介绍。然后在 6.2.3 小节和 6.3.1 小节结合二叉树，对函数重载做进一步的深入介绍。函数模板在本书的第 9 章排序中才正式引入，排序算法的实现从一般函数过渡到了模板函数，以提高学生的抽象认识。

对于广义表类，为实现其遍历等操作，首先需解决广义表的创建操作。本书采用简明方式实现了广义表的字符逐一读入和在内存中建立起该表。

关于图的存储结构（邻接矩阵和邻接表）在现有的数据结构教材中，大多采用体系完整的、复杂的描述方法，教与学具有一定难度。在图类设计中，本书给出了简明的存储描述作为数据成员，提供了体现重要算法的较少函数成员，以便突出核心概念和重点。对于图的遍历算法，书中给出了基于两种存储结构的 4 种遍历函数，使抽象的算法显得具体可行。对于图的应用中的复杂典型算法，为了突出思想和方法，并未采用类设计而以一般函数的形式给出。

本书作者曾于 2004 年将“数据结构”课程建设成浙江省精品课程，在课程网站（http://sjjg.js.zwu.edu.cn/）上提供了丰富的教学资源。

本书可以作为普通高等院校计算机专业本科、专升本教材。同时，也可作为研究生入学考试和各类认证证书考试的复习参考书，还可供计算机应用工程技术人员学习参考。

本书由杨秀金主编。其中第 1、2、5、6、8、10、11 章由杨秀金编写；第 3、9 章由汪沁编写；第 4、7 章由刘晓利编写。全书由杨秀金统稿。

由于作者水平有限，书中难免存在不妥与疏漏之处，敬请广大读者批评指正。

杨秀金

2008 年 11 月

目　录

第1章 绪 论

随着计算机科学技术、计算机产业的迅速发展，计算机的应用普及也在以惊人的速度发展，计算机应用已经深入到人类社会的各个领域。计算机的应用早已不限于科学计算，而更多地应用在信息处理方面。计算机可以存储的数据对象不再是纯粹的数值，而扩展到了字符、声音、图像、表格等各种各样的信息。对于信息的处理也不再是单纯的计算，而是一些如信息存储、信息检索等非数值的计算。那么，现实世界的各种数据信息怎样才能够存储到计算机的内存之中，对存入计算机的数据信息怎样进行科学处理，这涉及计算机科学的信息表示和算法设计问题。为解决现实世界中某个复杂问题，总是希望设计一个高效适用的程序。这就需要解决怎样合理地组织数据、建立合适的数据结构，怎样设计适用的算法，以提高程序执行的时间效率和空间效率。“数据结构”就是在此背景下逐步形成、发展起来的。

主要内容：

- 数据结构的基本概念
- 抽象数据类型
- C++语言简介
- 算法描述与分析

1.1 问题的引入

1.1.1 引言

在各种高级语言程序设计的基本训练中，解决某一实际问题的步骤一般是：分析实际问题；确定数学模型；编写程序；反复调试程序直至得到正确结果。所谓数学模型一般指具体的数学公式、方程式等，如牛顿迭代法解方程，各种级数的计算等。这属于数值计算的一类问题。而现实生活中，更多的是非数值计算问题，如手机中的通讯录，人们对它的操作主要是查找、增加、删除或者修改电话记录。再如，人们经常在互联网上查阅各种新闻，或查阅电子地图，人们可以在某城区地图上查找自己所需的街道或店铺，其操作主要是搜索和查询。下面再来分析几个典型实例，它们的主要特点是：不同实例的数据元素之间存在不同的关系；对数据信息的处理主要有插入、删除、排序、检索等。

首先分析图书目录卡或学籍档案类问题，手机通讯录也属同一类问题。设一个班级有30个学生，其学籍表如图1.1所示。

这个班级有30个学生。对应于该学籍表，它由30个记录组成，所谓记录也称为数据元素。先把每个学生的信息看成一个记录（即数据元素），表中的每个记录又由7个数据项组成。

在学籍表中 30 个数据元素之间是一种顺序关系。第 1 个学生的信息后面是第 2 个学生的信息……第 29 个学生后面是第 30 个学生的信息，数据之间的这种顺序关系称为线性关系。因此，通常这类表被称为线性表，数据之间的逻辑结构称为线性结构，其主要操作有查找、插入、删除等。

序号	学号	姓名	性别	英语	数学	物理
01	070301	李晓明	男	86	91	80
02	070302	张国庆	男	76	83	85
⋮	⋮					⋮
30	070330	王薇薇	女	88	93	90

图 1.1　学籍表

现在再来分析棋类对弈问题，与之相类似的还有体育竞技比赛问题等。在体育竞技中，先从基层的预选比赛开始，获胜者再继续参加比赛，最后决赛出一个冠军。在此，可把预选赛的参加者看成树叶，比赛最后所产生的冠军看成一个树根，这样形成一个树结构。再如，家族图谱问题，如果把祖父看成树根，把他的儿女们看成树的中间结点，再把祖父的孙子们看成树叶结点，这样也形成一个树结构。又如，可以把一所高校名称看成树根，把下设的若干个二级学院名以及每个二级学院下设的若干专业系名看成树的中间结点，最终把每个系所分出的若干专业方向看成树叶，这样还是一个树结构。理工学院的专业设置树结构如图 1.2 所示。

树结构的实例还有很多，如计算机软件系统的文件目录等。在树结构中的每个结点代表一个数据元素（可以包含较多的数据项），结点之间的关系不再是顺序的，而是分层、分叉（互不相交）的树结构。对于树结构的主要操作是遍历、查找、插入、删除等。

最后分析交通或通信网问题。如果把若干个城镇看成若干个顶点，再把城镇之间的道路看成边，它们可以构成一个网状的图，如图 1.3 所示。

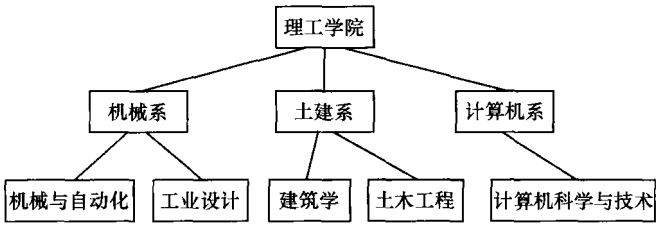


图 1.2　专业设置树

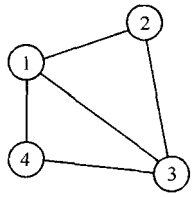


图 1.3　交通示意图

图中每个顶点通常视作一个数据元素，图 1.3 中的一个顶点信息可以代表一个城镇的主要简况，各个顶点之间的关系由边来表示，它可以视作城镇间的道路或通信线缆等。数据元素之间的关系纵横交错，更加复杂，这种结构被称为图或网状结构。在实际应用中，假设某地区有 4 个城镇，有一调查小组要对该地区每个城镇进行调查研究，并且每个城镇仅能调查一次，试问调查路线怎样设计才能以最高的效率完成此项工作？这就是一个图论方面的问题。这类问题的处理，有最短路径问题、关键路径问题、地图染色、背包等问题。它的存储和管理不属于单纯的数值计算问题，而是一种复杂的非数值信息处理问题。

在上述实例中，还反映出不同问题的数据元素之间的逻辑关系也各不相同。

1.1.2　数据结构课程研究内容

在计算机学科中，“数据结构”是研究一类数据的表示及其相关的运算的一个专门领域。

“数据结构”主要研究怎样合理地组织数据、建立合适的数据结构、设计适用的算法，以提高计算机执行程序所用的时间效率、空间效率等问题。可以概括为“数据结构”主要研究：数据之间的逻辑结构->数据在机内的存储结构->相关运算以及分析。

1968 年，美国的 D.E.Knuth 教授开创了“数据结构”的最初体系，他的名著《计算机程序设计技巧》第一卷《算法基础》较为系统地阐述了数据的逻辑结构和存储结构及其操作。随着计算机科学的飞速发展，到 20 世纪 80 年代初期，“数据结构”的基础研究已日臻成熟。

“数据结构”是计算机专业的一门综合性专业基础课。它以数学为基础，所研究的内容不仅涉及计算机硬件的研究范围，而且与计算机软件的研究有着更加密切的关系。“数据结构”以高等数学、线性代数、离散数学、概率与数理统计为数学基础，还以某高级程序设计语言为程序设计基础。经过本课程的学习训练，它可为操作系统、数据库原理、编译原理等后续专业课程的学习奠定基础。“数据结构”涉及各方面的知识，如计算机硬件研究范围的存储装置、存取方法和编码理论，在计算机软件研究范围的文件系统、数据的动态存储与管理、信息检索等，还涉及一些综合性的知识，如数据类型、数据表示、数据运算、数据存取等方面的知识，因此，“数据结构”是涉及数学、计算机硬件和软件 3 个范畴的一门核心课程，如图 1.4 所示。

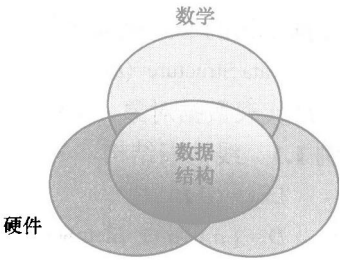


图 1.4　“数据结构”是核心课程

“数据结构”课程本身除了具有一定的理论性、抽象性和复杂性之外，又有很强的实践性、应用性。由于数据结构是计算机各种大程序的中枢和骨架，因此，它不仅是计算机专业的核心课程，而且也是电子信息以及其他相关专业的重要选修课程之一。

1.2　数据结构的基本概念

1.2.1　计算机领域中的数据

在计算机科学中，**数据**（Data）是描述客观事物的数字、字符以及所有能够输入到计算机中并被计算机处理的信息的总称。除了数字、字符之外，还有用英文、汉字或其他语种字母组成的词组、语句。此外，图形、图像、声音等信息也可称之为数据。

数据元素（Data Element）是数据的基本单位，在计算机中通常作为一个整体进行考虑和处理，如图 1.2 所示的“专业设置树”中的一个专业，图 1.3 所示的“交通图”中的一个城镇都可称为一个数据元素。数据元素除了可以是一个数字（最简单情况）或一个字符串以外，它也可以由一个或多个数据项组成，如图 1.1 中所示每个学生的学籍信息作为一个数据元素，在表中占一行，每个数据元素由序号、学号、姓名、性别、英语成绩等 7 个数据项组成。**数据项**（Data Item）是有独立含义的数据的最小单位，数据项有时也称为**字段**（Field）。

数据对象（Data Object）是具有相同性质的数据元素的集合，是数据的一个子集。例如，整数数据对象是集合 $N=\{0, \pm1, \pm2, \cdots\}$，大写英文字母字符数据对象是集合 $C=\{'A', 'B', \cdots, 'Z'\}$。图 1.1 所示的学籍表，也是一个数据对象。

1.2.2 数据结构相关概念

在前文对数据结构的介绍中，已经提到一些术语，如逻辑结构、存储表示、运算处理等。下面将对其进行严格的定义和阐述。

数据的逻辑结构（Data Structure）是带有结构的数据元素的集合。它是指数据元素之间的相互关系，即数据的组织形式。数据的逻辑结构是针对多个数据元素而言的。把数据元素之间的逻辑上的联系称之为数据的逻辑结构，如前文提到的线性结构、树结构、图或网状结构。在人类历史发展过程中，“数据的逻辑结构”早就客观存在，如部落群组、家族图谱属于树结构，古代、近代战争的作战图就是图结构。数据的逻辑结构体现数据元素之间的抽象化相互联系，逻辑结构并不涉及数据元素在计算机中具体的存储方式，是独立于计算机的。

数据结构的形式化描述是：

Data Structure=(D, R)

其中，D 代表数据对象，R 代表数据关系。

例 1.1 线性表结构

List = (D,R)

D = { a_i| $a_i \in$ ElemSet, i = 1,2,⋯, n, $n \geq 0$}

R = { $<a_{i-1},a_i>$ | $a_{i-1},a_i \in$ D, i=2,3,⋯, n}

其中，D 代表数据对象，如果 ElemSet 具体化为整型 int，上述 D 代表有 n 个整型数据的集合。R 代表数据关系，体现数据元素之间的逻辑结构。所有数据元素存在次序关系$<a_{i-1},a_i>$，每两个相邻的数据元素互为前驱和后继关系，a_1 无前驱，a_n 无后继。这种关系称为线性关系，这里描述的就是线性表。

例 1.2 复数结构

Complex = (D,R)

D = { c_1,c_2 | c_1,c_2 ∈ float }

R = { $<c_1,c_2>$ }　　c_1,c_2 分别代表复数的实部和虚部

其中，D 代表数据对象，仅有两个实数；R 代表数据关系，体现数据之间的逻辑结构；c_1 ,c_2 分别代表复数的实部和虚部。这里描述的是复数结构。

例 1.3 树型结构

某课题小组有 1 位教师 T，3 名研究生 G，6 名本科生 S。他们的关系是，教师指导 3 名研究生，每个研究生指导 2 名本科生。

Group = (D,R)

D = { T,G_i,S_{ij},　i=1,2,3,　j=1,2,}

R = { $R1$,$R2$ }

$R1$ = {$<T, G_i>$　　i=1,2,3,}

$R2$ = {$< G_i, S_{ij}>$　i=1,2,3,　j=1,2,}

根据对课题研究小组的形式化描述，可以画出如图 1.5 所示的树型结构。

讨论数据结构的目的是为了在计算机中实现对大量数据信息的操作处理，因此，还需要研究如何在计算机中存储和表示数据信息。计算机的内存具有一定的容量，常见的有几百

MB，到几个 GB，如 512MB 或 2GB 等。内存中每个存储单元的地址是顺序排放的。对于线性表，数据的逻辑结构本身就是顺序的，它的存储方式也可以是顺序的。那么，树、图结构的数据信息在内存之中又如何存储呢？数据的逻辑结构在计算机存储设备中的映像被称为数据的存储结构，也可以说**数据的存储结构**是逻辑结构在计算机存储器里的实现，又称为物理结构。数据的存储结构是依赖于计算机的，常见的存储结构有顺序存储结构（顺序映像）、链式存储结构（非顺序映像）等。

顺序存储结构如图 1.6 所示。数据元素在内存中存放的地址是连续的。

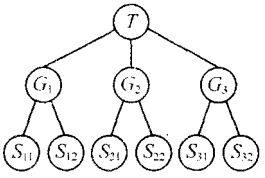


图 1.5　课题小组成员间的树型关系

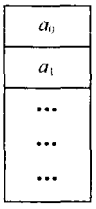


图 1.6　顺序存储结构

链式存储结构如图 1.7 所示。其主要特点是每个数据元素在内存中存放的地址可以是不连续的。每个数据元素所占用的存储单元除了存放数据元素本身，同时必须存储下一个数据元素所占用存储单元的地址（也称指针）。假设有一线性表，它仅有 3 个数据元素（a_0，a_1，a_2）。a_0 和 a_1 它们在逻辑关系上是相邻元素，而在计算机中存放的物理位置不一定是连续的。在图 1.7 中，a_0 和 a_1 的物理位置就是不连续的（不相邻）。a_0 后面的指针指向 a_1 地址；a_1 后面的指针指向 a_2 地址；而 a_2 后面的指针为空（在图中用^表示），表明 a_2 是最后一个数据元素。图 1.7（a）所示为链式结构在内存中的情况，图 1.7（b）所示为链式结构的常用表示图。

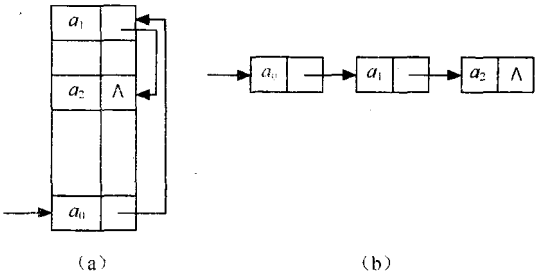


图 1.7　链式存储结构

关于链表的详细解释将在第 2 章中介绍。

通常所谓“数据结构”是指数据结构这门课程，它的研究内容包括数据的逻辑结构、数据在计算机内的存储结构以及定义在它们之上的一组运算。这里的运算实质是对数据的处理，处理数据的思路和方法常称之为**算法**。算法的基本概念在本章 1.5 节介绍。不论是存储结构的设计，还是运算的算法设计，都必须考虑存储空间的开销和运行时间的效率，因此，“数据结构”课程不仅讲授数据信息在计算机中的组织和表示方法，同时也训练高效地解决复杂问题进行程序设计的能力。

1.3　抽象数据类型

1.3.1　数据类型

数据类型（Data Type）是和数据结构密切相关的一个概念，它最早出现在高级语言中。

在高级语言的程序设计中，每个常量、变量或表达式都有确定的数据类型。**数据类型**实质上是一组值的集合和定义在该集合之上的一组操作的总称。

例如，在C/C++语言中的整数类型（int）就是一个最常用的数据类型，它的数值集合范围是[-32768，32767]，主要运算有：+、-、*、/、%（取模运算）等。从硬件角度看，它们的实现会涉及“字”、“字节”、“位”、“位运算”等。但从用户角度看，并不需要了解整数在计算机内是如何表示的，也不需要了解运算细节是如何实现的。用户只需了解整数的加法、减法、取模运算等的抽象特性，而不必了解相应的“位运算”细节，就可运用高级语言进行程序设计。

再如，C++语言中的布尔类型（bool），它的值集范围是[true,false]，主要运算有：!（非）、&&（与）、||（或）等。同样，用户只需了解布尔类型的非、与、或运算的抽象特性，而不必了解在计算机内的实现细节，就可在程序中加以运用。

从事计算机专业的人员，需要对“抽象”有清楚的理解和认识。

1.3.2 抽象数据类型

抽象数据类型（Abstract Data Type，ADT）是指基于一类逻辑关系的数据类型以及定义在这个类型之上的一组运算。抽象数据类型的定义取决于客观存在的一组数据的逻辑特性（即数据元素之间的逻辑关系，如线性、树、图等关系），而与其在计算机内如何表示和实现无关，即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部使用。在某种意义上讲，抽象数据类型和数据类型实质上是一个概念。整数类型就是一个简单的抽象数据类型实例，“抽象”的意义在于数学特性的抽象。另一方面，抽象数据类型的含义更广，不仅限于各种不同的计算机处理器中已定义并实现的数据类型，如字符型、整型、实型等，还包括设计软件系统时用户自己定义的复杂数据类型。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复用程度就越高。

抽象数据类型的描述可以归纳为：

```
ADT name{
    数据对象：D={ …… }
    数据关系：R={ …… }
    基本操作：创建；
              输出；
              插入；
              删除；
              等等；
  } ADT name;
```

首先回顾，一个线性表的一般化简单表示为：$(a_1, a_2, \cdots, a_{n-1}, a_n)$。线性表的形式化表示为：List = (*D*,*R*)，详见1.2.2小节的例1.1。

那么，线性表的抽象数据类型描述如下：

```
ADT  Linear_list {
    数据对象：D={a_i |  a_i∈ ElemSet , i=1.2,…,n    n≥0 ; }
    数据关系：R={ <a_i-1 ,a_i > | a_i-1 ,a_i∈≥D, i=2,3,…,n }
```

基本操作： 初始化空线性表;
求线性表表长;
取线性表的第 i 个元素;
在线性表的第 i 个位置插入元素 x;
删除线性表的第 i 个元素;
等等;
} **ADT**　Linear_list;

上述 ADT 很明显是抽象的，数据元素所属的数据对象没有局限于一个具体的整型、实型或其他类型。所具有的操作也是抽象的数学特性，并没有具体到何种计算机语言指令与程序编码。可以看出，线性表抽象数据类型的描述与线性表的形式化定义描述（List=(*D*,*R*)）十分相似，只是前者 ADT 比后者增加了对数据对象的若干操作的描述。

“数据结构”课程，既讨论抽象的 ADT，又讨论怎样在计算机中具体实现 ADT。在本书第 2 章有对 ADT Linear_list 实现的详细介绍。

现在来看复数抽象数据类型的描述：

ADT complex{
数据对象： D={ c_1,c_2　|　c_1,c_2∈FloatSet }
数据关系： R={ <c_1,c_2>　c_1,c_2 分别为实部、虚部　}
基本操作： 创建一个复数;
输出一个复数;
求两个复数相加之和;
求两个复数相减之差;
求两个复数相乘之积;
等等;
} **ADT** complex;

对于抽象数据类型的具体实现，依赖于所选择的高级语言的功能。从程序设计的历史发展来看，有如下几种不同的实现方法。

第 1 种是传统的面向过程的程序设计，这种方法长期以来作为程序设计的主流方法，现在仍然是程序设计入门和基础教育的主流方法。它根据数据的逻辑结构选定合适的存储结构，根据所要求的操作设计出相应的子程序或子函数，一个源程序由若干个函数构成。当前广泛使用的 C 语言就是一种典型的面向过程语言。

第 2 种是“包”、“模块”的设计方法，将这种方法看成是一种过渡或许更确切。Ada 语言提供了“包”（Package）结构，Module-2 语言提供了“模块”（Module）结构，TURBO PASCAL 语言提供了“单元”（Unit）结构，用这类结构实现 ADT 比起第 1 种方法有一定的进步。

第 3 种是面向对象的程序设计（Object Oriented Programming，OOP），这种方法正在逐步成为当今程序设计的主流方法。在面向对象的程序设计语言中，存储结构和操作函数的说明被封装在一个整体结构中。存储结构的说明称为它的数据成员，操作函数的说明称为它的函数成员，这个整体结构称之为“类”（Class），属于某个“类”的具体变量称之为“对象”（Object）。用面向对象的程序设计实现 ADT，两者更加接近和一致。本书将以面向对象程序设计为主进行讨论。

1.3.3 抽象数据类型的实现

对于前面介绍的复数抽象数据类型 ADT complex，现在来研究它的两种不同的实现方法。

例 1.4 复数的抽象数据类型（ADT），面向过程的程序实现。

[C 语言源程序]

```
#include <stdio.h>
typedef struct                    /* 存储表示，结构体类型的定义 */
        { float   x;              /*  实部子域 */
          float   y;              /* 虚部的实系数子域  */
        }comp;
/* 子函数的原型声明 */
void creat(comp *c);
void outputc(comp a);
comp add(comp k,comp h);
comp sub(comp k, comp h):
comp a,b,a1,b1;   int z;          /* 全局变量的说明 */
int main()                        /* 主函数 */
{   creat(&a);     outputc(a);
    creat(&b);     outputc(b);
    a1=add(a,b); outputc(a1);
    return 0;
}
void creat(comp *c)               /* 输入一个复数的实部和虚部  */
{ float x1,y1;
   printf("\n 输入实部 real x=?");scanf("%f",&x1);
   printf("\n 输入虚部 xvpu y=?");scanf("%f",&y1);
   c->x=x1;   c->y=y1;
}
void outputc(comp a)              /*  输出一个复数  */
 { printf("\n   复数：%f + %f*i",a.x,a.y) ;
   }
comp add(comp k,comp h)           /*  求两个复数相加之和 */
{ comp l;
   l.x=k.x+h.x;   l.y=k.y+h.y;
   return l;
}
comp sub(comp k,comp h)           /* 求两个复数相减之差  */
{ comp l;
```