21 面向CENTURY 高等学校计算机类专业规划教材

数据结构教程—Java语言描述

朱振元　朱　承　刘　聆　编著

西安电子科技大学出版社
http://www.xduph.com

面向21世纪高等学校计算机类专业规划教材

数据结构教程

——Java语言描述

朱振元　朱　承　刘　聆　编著

西安电子科技大学出版社

2007

内 容 简 介

本书采用面向对象的 Java 语言实现抽象数据类型，对每一种抽象的数据类型先定义成接口，然后再结合各种具体的存储结构加以实现，并以各实现类为线索对各种操作的实现方法加以说明。其内容包括：线性表、栈、队列、串、数组、矩阵、集合、广义表、树、图等数据结构及查找和排序的方法。本书突出应用性，在书中除列举算法设计实例外，还使用开发工具 JBuilderX 对典型的应用问题加以实现。

本书语言通俗，条理清晰，应用性强，可作为高等院校计算机专业的教科书，也可作为应用程序开发人员及电脑爱好者的技术参考书。

★本书配有电子教案，需要的老师可与出版社联系，免费提供。

图书在版编目(CIP)数据

数据结构教程：Java 语言描述 / 朱振元，朱承，刘聆编著. —2 版.
—西安：西安电子科技大学出版社，2007.12
面向 21 世纪高等学校计算机类专业规划教材
ISBN 978-7-5606-0895-2
Ⅰ. 数… Ⅱ. ①朱… ②朱… ③刘… Ⅲ. ①数据结构—高等学校—教材
②JAVA 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(2007)第 159780 号

责任编辑 寇向宏 云立实
出版发行 西安电子科技大学出版社(西安市太白南路 2 号)
电 话 (029)88242885 88201467 邮 编 710071
http: //www.xduph.com E-mail: xdupfxb@pub.xaonline.com
经 销 新华书店
印刷单位 陕西天意印务有限责任公司
版 次 2007 年 12 月第 1 版 2007 年 12 第 1 次印刷
开 本 787 毫米×1092 毫米 1/16 印 张 22
字 数 522 千字
印 数 1～4000 册
定 价 29.00 元
ISBN 978 - 7 - 5606 - 0895 - 2/TP • 0477
XDUP 1166011-1

前　言

数据结构课程是我国计算机教学中较早形成和完善的一门专业基础课程，也是计算机课程体系中的核心课程之一。在该课程中所介绍的各类数据的逻辑结构、储存方式及相关的算法既是程序设计(特别是非数值性程序设计)的基础，又是设计和实现系统软件及大型应用软件的重要基础。

随着计算机技术的飞速发展，程序设计方法及软件开发技术也出现了重大的发展及变革。目前，面向对象的软件分析和设计技术已发展成为软件开发技术的主流。

Java 是一种纯粹的面向对象的程序设计语言，它是在 C++的基础上发展起来的，继承了 C++中大量有效的语法成分，但也对 C++作了不少改进，抛弃了 C++中容易引起问题的成分。同时，Java 语言又是一种网络程序设计语言，它所具有的平台无关性可以使以这种语言编写的程序能不作修改地在任何一台计算机上正确地运行。Java 语言的这些特点使它已经成为目前主流的开发语言。作为一种通用语言，Java 语言几乎是完美的。因此，本书选择 Java 语言作为描述语言。

本书以培养与提高学生的基本专业素质及综合应用能力为追求目标，注重体现以下特色:

(1) 先进性。采用面向对象的观点讨论数据结构技术，对每一种抽象的数据类型先定义成接口，然后再结合各种具体的存储结构加以实现，并以各实现类为线索对类中各种操作的实现方法加以说明。与传统的实现方法相比，采用这种方法可使程序具有更高的可靠性、可维护性和可复用性。

(2) 应用性。在本书中设置了大量的应用实例，这些实例分为“算法设计”与“综合应用”两个层次。“算法设计”一般是作为类的成员函数在相关的类中介绍，而“综合应用”程序在每一章的最后一节介绍，内容包括各章所介绍的类的功能演示及典型的应用问题，并使用面向对象的开发工具 JBuilderX 来实现这些综合应用程序。

(3) 适应性。在本书中对每一种数据类型都有比较规范的表述过程，对每一种算法都有规范统一的说明步骤，对算法的含义、参数与功能、工作变量、处理过程都进行了明确的说明，并通过图示、文字注释、实例的执行过程等多种方式来帮助学生理解算法，提高学生的计算思维能力。

本书在《数据结构——C++语言描述》(清华大学出版社，2007 年)的基础上改编而成。除结合 Java 语言的特点对原书作了一系列改进之外，还对章节的设置作出了合理的调整，删除了原书中的最后一章“外部排序”，而将“递归算法”单独作为一章。

由于考虑到读者应具有Java语言的基础知识，因此在附录中给出了Java语言的概要，并对Java语言与C++语言在使用方面的特点进行了比较。

本书主要由朱振元、朱承、刘聆编著，何文德、刘钢钦、欧阳实三位老师对全书的内容提出了许多宝贵的意见，并参与了部分编写工作。本书的改编还得到了许多人士的帮助，在此一并表示诚挚的谢意。

由于本书在总体策划及实现方法方面都作了一些新的尝试，加之作者水平有限、时间仓促，因此书中难免存在缺点与疏漏，敬请读者及同行予以批评指正。

朱 振 元

2007年4月

目　录

第 1 章　课程概论 1
1.1　课程的初步认识 1
1.2　数据结构的基本概念 3
1.2.1　基本术语 3
1.2.2　数据结构的概念 3
1.2.3　逻辑结构和物理结构 4
1.2.4　数据结构形式的定义 4
1.3　数据类型及面向对象的概念 5
1.3.1　数据类型概述 5
1.3.2　抽象数据类型 6
1.3.3　实现方法 6
1.3.4　面向对象的概念 8
1.4　算法及算法分析 9
1.4.1　算法特性 10
1.4.2　算法描述 10
1.4.3　算法的设计要求 11
1.4.4　算法分析 11
习题 1 14
第 2 章　线性表 15
2.1　线性表的相关概念及抽象数据类型 15
2.1.1　线性表的相关概念 15
2.1.2　线性表抽象数据类型描述 16
2.1.3　线性表的接口定义 17
2.2　顺序表类 18
2.2.1　顺序表的存储结构 18
2.2.2　顺序表的类定义 19
2.2.3　顺序表类的实现 21
2.2.4　顺序表算法设计举例 24
2.3　单链表类 27
2.3.1　单链表的存储结构 27
2.3.2　单链表的类定义 28
2.3.3　单链表类的实现 30
2.3.4　单链表算法设计举例 34
2.4　双向循环链表类 35
2.4.1　双向循环链表的存储结构 35
2.4.2　双向循环链表的类定义 36
2.4.3　双向循环链表类的实现 38
2.5　线性表应用举例 42
2.5.1　线性表功能演示程序 42
2.5.2　进程调度模拟程序 44
习题 2 49
第 3 章　栈 50
3.1　栈的相关概念及抽象数据类型 50
3.1.1　栈的相关概念 50
3.1.2　栈抽象数据类型描述 51
3.1.3　栈的接口定义 52
3.2　顺序栈类 52
3.2.1　顺序栈的存储结构 52
3.2.2　顺序栈的类定义及实现 53
3.2.3　顺序栈算法设计举例 55
3.3　链栈类 56
3.3.1　链栈的存储结构 57
3.3.2　链栈的类定义及实现 57
3.3.3　链栈算法设计举例 60
3.4　栈的应用举例 61
3.4.1　括号配对问题 61
3.4.2　表达式求值 63
3.4.3　栈功能演示程序 68
习题 3 71
第 4 章　队列 73
4.1　队列的相关概念及抽象数据类型 73
4.1.1　队列的相关概念 73
4.1.2　队列抽象数据类型描述 74
4.1.3　队列的接口定义 74
4.2　链队列类 75
4.2.1　链队列的存储结构 75

4.2.2 链队列的类定义及实现 76
4.2.3 链队列算法设计举例 79
4.3 循环队列类 80
4.3.1 队列的顺序存储结构 80
4.3.2 循环队列类的定义及实现 83
4.3.3 循环队列算法设计举例 85
4.4 队列的应用举例 87
4.4.1 显示杨辉三角形 87
4.4.2 表达式(逆波兰表示)求值 90
4.4.3 循环队列功能演示程序 93
习题4 96
第5章 串 97
5.1 串的相关概念及抽象数据类型 97
5.1.1 串的相关概念 97
5.1.2 串抽象数据类型描述 98
5.1.3 串的接口定义 99
5.2 串的存储结构 100
5.2.1 顺序存储结构 100
5.2.2 链式存储结构 101
5.2.3 堆存储结构 101
5.3 串值不变的顺序串类 Str1 101
5.3.1 Str1 的类定义 102
5.3.2 求子串、定位操作的实现 103
5.3.3 删除、插入及替换操作的实现 106
5.3.4 Str1 类算法设计举例 108
5.4 串值可变的顺序串类 Str2 110
5.4.1 Str2 类的定义 110
5.4.2 Str2 类的实现 110
5.4.3 Str2 类算法设计举例 112
5.5 串的应用举例 113
5.5.1 字符串类功能演示程序 113
5.5.2 文本文件单词统计程序 115
习题5 117
第6章 数组、矩阵和集合 118
6.1 数组 118
6.1.1 数组的相关概念及抽象数据类型 118
6.1.2 数组类的定义及实现 119
6.2 矩阵概述 121
6.2.1 矩阵的相关概念 121
6.2.2 矩阵的存储方式 122
6.3 矩阵类 123
6.3.1 矩阵类的定义 123
6.3.2 矩阵类的实现 124
6.3.3 矩阵类算法设计举例 127
6.4 矩阵的压缩存储 128
6.4.1 对称矩阵的压缩存储 128
6.4.2 对角矩阵的压缩存储 129
6.4.3 稀疏矩阵的压缩存储 130
6.5 稀疏矩阵类 133
6.5.1 稀疏矩阵类的定义 133
6.5.2 稀疏矩阵类的实现 134
6.5.3 稀疏矩阵类算法设计举例 137
6.6 集合 138
6.6.1 集合的相关概念及抽象数据类型 138
6.6.2 整数集合类 139
6.7 应用举例 142
6.7.1 在二维数组中存储对称矩阵 142
6.7.2 整数集合类应用程序 143
习题6 144
第7章 递归算法 146
7.1 递归的概念 146
7.2 递归函数的设计方法 148
7.2.1 由递归定义设计递归函数 148
7.2.2 回溯法求解的一般模式 150
7.3 算法设计举例 150
7.4 递归算法的非递归化实现 153
7.4.1 自底向上的递推 153
7.4.2 为递归参数设置循环 153
7.4.3 自顶向下的递推 154
7.4.4 使用无标号递归工作栈 155
7.5 递归算法应用举例 157
7.5.1 方螺旋线求解程序 157
7.5.2 八皇后问题求解程序 160
7.5.3 迷宫问题演示程序 162
习题7 166
第8章 广义表 167
8.1 广义表的相关概念及抽象数据类型 167
8.1.1 广义表的相关概念 167

8.1.2 广义表抽象数据类型描述 168
8.2 广义表的存储方式 169
8.2.1 头尾表示法 169
8.2.2 儿子兄弟表示法 171
8.3 广义表类的定义及实现 172
8.3.1 广义表类的定义 172
8.3.2 建立广义表的存储结构 175
8.3.3 取头、取尾操作的实现 178
8.3.4 插入、删除操作的实现 179
8.4 广义表的递归算法 179
8.4.1 广义表的相等比较 179
8.4.2 广义表的成员判别 180
8.4.3 求广义表的深度 181
8.4.4 广义表递归算法设计举例 182
8.5 广义表应用举例 185
8.5.1 广义表演示程序 185
8.5.2 LISP 表达式求值 187
习题 8 189
第 9 章 树与二叉树 190
9.1 树的相关概念及抽象数据类型 190
9.1.1 树的定义 190
9.1.2 树的逻辑表示 191
9.1.3 基本术语 192
9.1.4 树抽象数据类型描述 193
9.2 二叉树 194
9.2.1 二叉树的定义及抽象数据类型 194
9.2.2 二叉树的基本性质 194
9.2.3 二叉树的存储结构 196
9.2.4 二叉树类的定义 198
9.2.5 二叉树类的实现 201
9.2.6 二叉树的遍历 204
9.2.7 二叉树算法设计举例 208
9.3 排序二叉树 209
9.3.1 排序二叉树的定义 209
9.3.2 排序二叉树类的定义 209
9.3.3 排序二叉树类的实现 210
9.4 树与森林 211
9.4.1 树的存储结构 211
9.4.2 森林与二叉树的转换 213
9.4.3 树的遍历 214
9.5 哈夫曼树 214
9.5.1 哈夫曼树的定义 215
9.5.2 哈夫曼树的构造 216
9.5.3 哈夫曼编码 216
9.6 树的应用举例 218
9.6.1 哈夫曼编码生成程序 218
9.6.2 二叉树遍历演示程序 221
习题 9 224
第 10 章 图 226
10.1 图的相关概念及抽象数据类型 226
10.1.1 图的定义 226
10.1.2 基本术语 228
10.1.3 图抽象数据类型描述 230
10.2 图的存储方式 230
10.2.1 邻接矩阵 230
10.2.2 邻接链表 232
10.2.3 邻接多重表 234
10.3 图的遍历 235
10.3.1 邻接链表图类 235
10.3.2 深度优先搜索遍历 237
10.3.3 广度优先搜索遍历 238
10.3.4 算法设计举例 240
10.4 图的应用 242
10.4.1 拓扑排序 242
10.4.2 最短路径 246
10.5 图的应用程序设计举例 249
10.5.1 图的遍历演示程序 249
10.5.2 最短路径应用程序 253
习题 10 254
第 11 章 查找 256
11.1 查找的相关概念 256
11.2 静态查找表 257
11.2.1 顺序表的查找 257
11.2.2 有序表的查找 260
11.2.3 静态树表的查找 263
11.2.4 索引顺序表的查找 266
11.3 动态查找表 267
11.3.1 排序二叉树的查找 267

11.3.2 B−树与 B+树 272
11.4 哈希表 277
11.4.1 哈希表的概念 277
11.4.2 常用哈希函数 279
11.4.3 冲突的处理方法 281
11.4.4 哈希表的查找 283
11.5 查找应用程序举例 284
11.5.1 排序二叉树演示程序 284
11.5.2 图书信息查询程序 285
习题 11 289
第 12 章 排序 291
12.1 排序的相关概念 291
12.2 几种简单的排序算法 293
12.2.1 直接插入排序 293
12.2.2 冒泡排序 295
12.2.3 直接选择排序 297
12.3 几种快速的排序方法 300
12.3.1 快速排序 300
12.3.2 树型选择排序 303
12.3.3 堆排序 304
12.3.4 归并排序 309
12.4 基数排序 311
12.5 排序应用程序举例——排序算法演示程序 314
习题 12 317
附录 319
附录 A Java 语言概要(与 C++语言比较) 319
附录 B Java 语言中的常用类 323
附录 C JBuilderX 开发环境及操作步骤 329
附录 D 部分习题参考答案 333
参考文献 343

第1章 课 程 概 论

数据是信息的载体，随着信息化社会的发展，由计算机进行处理的数据量随之增大，数据类型随之增多，数据结构随之复杂。由于数据的组织方式直接关系到程序结构的优劣和程序处理的效率，因此这就给程序设计带来了一些新的问题。为了编出一个结构好、效率高的处理程序就必须分析待处理对象的特性，以及各处理对象之间存在的关系。这就是“数据结构”这门学科形成和发展的背景。

1.1 课程的初步认识

在计算机的使用初期，它的主要应用领域是科学计算。当人们使用计算机来解决一个具体问题时，一般需要经过下列几个步骤：首先，要从具体问题抽象出一个适当的数学模型；然后，设计或选择一个解此数学模型的算法；最后，编出程序，进行测试、调试，直至得到最终的解答。例如，求解梁架结构中的应力，其数学模型为线性方程组，可以使用迭代算法来求解线性方程组。

然而，随着计算机应用领域的不断扩大，许多具体问题已无法用数值及数学方程加以描述。这是一类非数值计算问题，下面所列举的就是属于这一类的具体问题。

【例 1.1】 人事信息检索问题。当我们要查找某公司员工的信息时，一般是给出该员工的编号或姓名，通过信息目录卡片和信息卡片来查得该员工的有关信息。但也有可能要查找某一种类别的员工信息，例如，我们要查找该公司的员工中具有“高级程序员”技术职称的人员信息，或者是属于某一行政分组的人员信息等。若利用计算机来实现人事信息检索，则计算机所处理的对象便是这些信息目录卡片及员工信息卡片中的信息。因此，在计算机中必须建立并存储与此相关的三张表，一张是按员工编号排列的员工信息表，另外两张分别是按技术职称与行政分组顺序排列的索引表。在员工信息表中存放编号、姓名、职称、职务及爱好等信息。在职称索引表中存放职称与员工编号的对应信息，在组别索引表中存放行政分组与员工编号的对应信息，如图 1.1 所示。

系统分析员	1
高级程序员	2,5,6
程序员	3,4,7,8,9,10

(a) 职称索引表

第一组	5,2,3,4
第二组	6,7,8,9,10

(b) 组别索引表

图 1.1 人事信息检索系统中的数据结构

在人事信息检索问题中，上述这几张表便是数学模型，计算机的主要操作便是按指定的要求对这些表进行查找。在这一类属于文档管理的数学模型中，计算机的处理对象之间通常存在着一种最简单的线性关系，相应的数据结构可称为线性数据结构。

【例 1.2】 八皇后问题。八皇后问题是求解在某些约束条件下，棋盘的合法布局。在八皇后问题中，处理过程不是根据某种确定的计算法则，而是利用试探和回溯的探索技术求解。为了求得合法布局，在计算机中要存储布局的当前状态。从最初的布局状态开始，一步步地进行试探，每试探一步形成一个新的状态，整个试探过程形成了一棵隐含的状态树，如图 1.2 所示(为了描述方便，我们将八皇后问题简化为四皇后问题)。回溯法求解过程实质上就是一个遍历这棵状态树的过程。在这个问题中所出现的“树”也是一种数据结构，它可以应用在许多非数值计算的问题中。

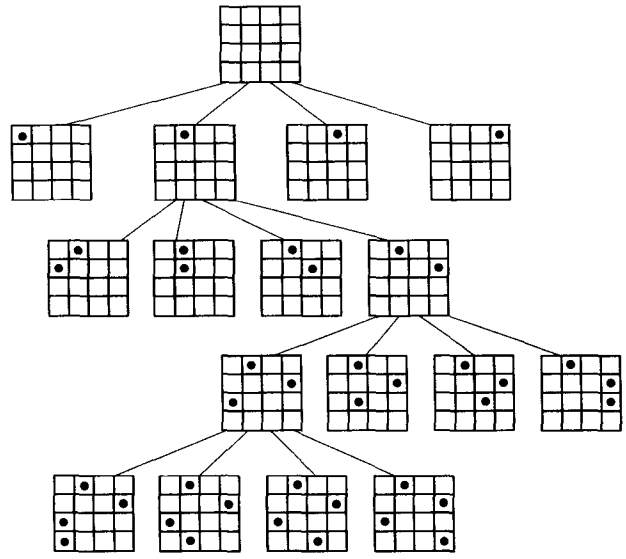


图 1.2　四皇后问题中隐含的状态树

【例 1.3】 交通咨询问题。交通咨询问题是求解两地间的最短路径或最少花费。在交通咨询问题中，可以采用一种图的结构来表示实际的交通网络，图中的顶点表示城市，边表示城市间的交通联系，对边所赋予的权值表示两城市间的距离，或途中所需的时间，或交通费用等。考虑到交通图的有向性(如航运、逆水和顺水时的船速就不一样)，图中的边可以用弧来表示，如图 1.3 所示。这个咨询系统可以回答旅客提出的各种问题，例如，从某地到某地应如何走才最节省费用。在这个问题中，所使用的图也是一种数据结构，它被用于解决这一类实际问题。

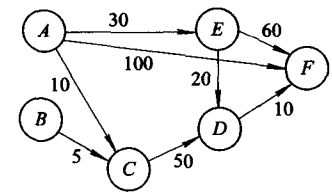


图 1.3　一个表示交通网的例图

综上三个例子可见，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树、图之类的数据结构。因此，可以说数据结构课程主要是研究非数值计算的程序设计问题中所出现的计算机操作对象以及它们之间的关系和操作的学科。

在计算机科学中，数据结构不仅是一般程序设计(特别是非数值计算的程序设计)的基础，而且是设计和实现编译程序、操作系统、数据库系统以及其他系统程序的大型应用程序的重要基础。

在计算机专业中，数据结构是一门综合性的专业基础课程，它不仅是计算机专业教学计划中的核心课程之一，而且是非计算机专业的主要选修课程。

学习数据结构课程的目的是为了了解计算机处理对象的特性，将现实世界的实际问题中所涉及的处理对象在计算机中表示出来并对它们进行处理。与此同时，通过算法训练以提高计算思维的能力，通过程序设计的技能训练来促进综合应用能力和专业素质的提高。

1.2 数据结构的基本概念

在系统地学习数据结构知识之前，我们先对一些概念和术语赋以确切的含义。

1.2.1 基本术语

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，一个利用数值分析方法解代数方程的程序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所使用的数据是字符串。随着计算机软、硬件技术的发展，及其应用领域的扩大，数据的含义也随之拓宽。像多媒体技术中所涉及的视频和音频信号，经采集转换后都能形成被计算机所接受的数据。

数据元素(Data Element)是数据的基本单位。在不同的条件下，数据元素又可称为元素、结点、顶点、记录。例如，在人事信息检索问题中员工信息表的一个记录，在八皇后问题中状态树的一个状态，在交通咨询系统中交通网的一个顶点等。在数据元素是记录的情形下，一个数据元素又可由若干数据项(也称为字段、域)组成，数据项是具有独立含义的最小单位。

数据元素类(Data Element Class)是具有相同性质的数据元素的集合。在某个具体问题中，数据元素都具有相同的性质(元素值不一定相等)，属于同一数据元素类，数据元素是数据元素类的一个实例。例如，在交通咨询系统的交通网中，所有的顶点是一个数据元素类，顶点 A 和顶点 B 各自代表一个城市，是该数据元素类中的两个实例，其数据元素的值分别为 A 和 B。

1.2.2 数据结构的概念

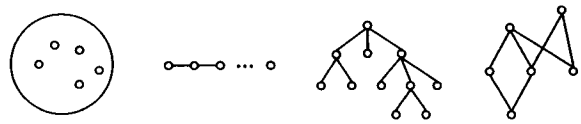
数据结构(Data Structure)是指互相之间存在着一种或多种关系的数据元素的集合。在任何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，这种数据元素之间的关系称之为结构。根据数据元素间关系的不同特性，通常有下列四类基本结构：

(1) 集合。在集合结构中，数据元素间的关系是“属于同一个集合”，集合是元素关系极为松散的一种结构。

(2) 线性结构。线性结构中的数据元素之间存在一个对一个的关系。

(3) 树形结构。树形结构中的数据元素之间存在一个对多个的关系。

(4) 图状结构。图状结构中的数据元素之间存在多个对多个的关系，图状结构也称为网状结构。图 1.4 为表示上述四类基本结构的关系图。



(a) 集合　(b) 线性　(c) 树形　(d) 图状

图 1.4　四类基本结构的关系图

1.2.3　逻辑结构和物理结构

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看做是从具体问题中抽象出来的数学模型，它与数据的存储无关。我们研究数据结构的目的是为了在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的表示(又称映像)称为数据的物理结构，或称存储结构。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

数据的存储结构可采用顺序存储或链式存储两种方法。

顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，元素间的逻辑关系由存储单元的相邻关系来体现。由此得到的存储表示称为顺序存储结构，顺序存储结构通常是借助于程序语言中的数组来实现的。

链式存储方法对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示。由此得到的存储表示称为链式存储结构，链式存储结构通常是借助于程序语言中的指针类型来实现的。

除了通常采用的顺序存储方法和链式存储方法外，有时为了查找的方便还采用索引存储方法和散列存储方法。

1.2.4　数据结构形式的定义

从上面所介绍的数据结构的概念中我们可以知道，一个数据结构有两个要素，一个是元素的集合，另一个是关系的集合。因此在形式上，数据结构通常可以用一个二元组来表示，即

$$\text{Data Structure} = (D, R)$$

其中，D 是数据元素的有限集，R 是 D 上关系的有限集。

例如，在上一节中所介绍的人事信息检索问题中，数据元素集合主要是员工信息表中的所有记录，而元素间关系的集合则可以从不同的视点去建立。我们可以按员工的编号来建立元素间的线性关系，从而形成线性的数据结构；可以按员工的行政分组来建立元素间的层次关系，从而形成树形的数据结构；可以按员工的爱好来建立元素间的网状关系，从而形成图状的数据结构，如图 1.5 所示。

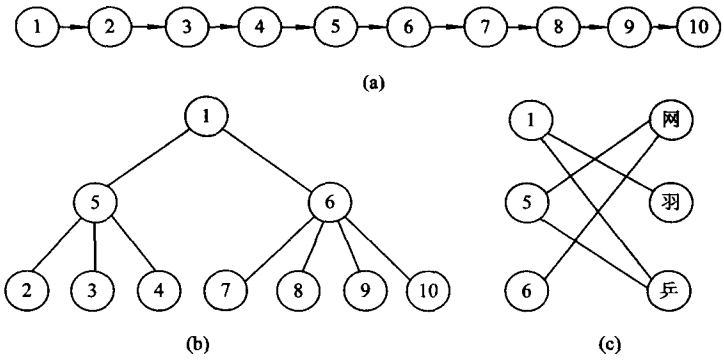


图 1.5 员工信息表中的数据结构

如果我们使用二元组来表示上述线性结构，则可表示为

$$\text{Linear-List} = (D, R)$$

其中：

D = {01,02,03,04,05,06,07,08,09,10} (用编号的后两位表示)

R = {<01,02>,<02,03>, …,<09,10>}

树形结构则可表示为

$$\text{Tree} = (D, R)$$

其中：

D = {01,02,03,04,05,06,07,08,09,10}

R = {<01,05>,<01,06>,<05,02>,…, 〈06,10>}

图状结构则可表示为

$$\text{Graph} = (D, R)$$

其中：

D = {01,02,03,04,05,06,07,08,09,10, 网，羽，乒}

R = {<01,羽>,<01,乒>,<05,网>,<05,乒>,<06,网>}

1.3 数据类型及面向对象的概念

1.3.1 数据类型概述

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。类型明显或隐含地规定了在程序执行期间变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。因此，数据类型是一个值的集合和定义在这个值集上的一组操作的总称。例如，PASCAL 语言中的整数类型，其取值范围为[−maxint,maxint]内的整数(maxint 是依赖特定计算机的最大整数)，定义在其上的一组操作为加、减、乘、除及取模运算等。

在高级程序语言中，数据类型可分为两类：一类是原子类型，另一类是结构类型。原

子类型的值是不可分解的，如 PASCAL 语言中的整型、实型等基本类型，而结构类型的值是由若干成分按某种结构组成的，因此是可分解的，并且它的成分可以是非结构的也可以是结构的。例如，数组的值由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，而数据类型则可看成是由一种数据结构和定义在其上的一组操作所组成。

1.3.2 抽象数据类型

抽象数据类型(Abstract Data Type，ADT)是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部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，都不影响其外部的使用。

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就是一个抽象数据类型，尽管它们在不同处理器上的实现方法可以不同，但因为其定义的数学特性相同，所以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。

但在另一方面，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，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的重用率，在近代程序设计方法学中要求在构成软件系统的每个相对独立的模块上，定义一组数据和施于这些数据上的一组操作，并在模块的内部给出这些数据的表示及其操作的细节，而在模块的外部使用的只是抽象的数据及抽象的操作。这也就是面向对象的程序设计方法。

抽象数据类型的定义可以由一种数据结构和定义在其上的一组操作组成，而数据结构又包括数据元素及元素间的关系，因此抽象数据类型一般可以由元素、关系及操作三种要素来进行定义。在本书中，对每一种数据类型都给出它的 ADT 定义。例如，栈的 ADT 定义：

元素　属于同一个数据元素类。

关系　数据元素间呈线性关系。

操作　Push(S, X)进栈操作，Pop(S)出栈操作等。

对于一个数据结构完全相同的数据类型，如果使它赋予不同的操作，则可形成不同的抽象数据类型。例如，将在第 3 章和第 4 章中介绍的栈和队列，它们可能都是相同的顺序表结构，但由于其操作不同，栈是先进后出，队列是先进先出，因而是属于两种不同的抽象数据类型。

抽象数据类型的特征是使用与实现相分离，实行封装和信息隐蔽。也就是说，在抽象数据类型设计时，把类型的定义与其实现分离开来。

1.3.3 实现方法

ADT 定义为数据类型建立了一个数学模型，其中包括数据结构及其相应的一组操作，而计算机上的具体实现则需借助于高级程序语言。在具体实现时大致可选择面向过程与面向对象两种不同的方法。

面向过程的方法着眼于系统要实现的功能。从系统的输入和输出出发，分析系统要做

哪些事情，进而考虑如何做这些事情，自上向下地对系统的功能进行分解，来建立系统的功能结构和相应的程序模块结构。但是，当程序因某种原因需要修改时，常常要涉及许多模块，有时因功能的改变而导致全部模块都要变更，这样的修改工作量极大并容易产生新的错误。

面向对象的方法着眼于应用问题中所涉及的对象，识别为解决问题所需的各种对象、对象的属性及相应的操作，从而建立起对象的类结构。通过对类的实体施行相应的操作以及各实体间的消息传递来实现系统的功能。类的定义充分体现了抽象数据类型的思想，基于类的体系结构可以把程序的修改局部化。当类中数据的存储方式及操作的实现过程需要修改时，不会影响外界对该类实体的操作，从而使整个系统保持稳定。因此，用面向对象开发方法建立起来的软件易于修改，与传统的方法相比，程序具有更好的可靠性、适用性、可修改性、可维护性、可复用性和可理解性。

下面我们就结合栈的演示程序的实现过程来比较这两种方法。虽然有关栈的内容我们还没有做过介绍，但从下面的比较中就可以大致领略到这两种程序设计方法的不同风格及各自的特点。

栈的特点是先进后出。这就像我们在生活中洗盘子一样，总是把洗好的盘子逐个放在其他已洗好的盘子的上面，而在使用盘子的时候，总是从上面逐个取出。一叠盘子相当于一个栈，放盘子的动作相当于进栈，取盘子的动作相当于出栈。如果要我们编制一个教学演示程序，模拟对栈进行入栈、出栈等操作的执行过程，则可以采用以下两种方法。

1．面向过程的方法

这是比较传统的方法，在这种方法中，可使用类型定义来描述其存储结构，并可借助于函数与过程描述其相应的操作。但在这两者之间并没有建立必然的联系。以下是使用 C 语言编写的程序代码，在程序中用一个字符表示栈中的一个元素，输入一个字符，若为“P”则表示执行出栈操作，若为“E”则表示退出执行，否则将该字符推入栈中。程序从开始起顺序地执行直至输入字符“E”。

```
Node* top; char ch;
void push(char ch);
⋮
char pop( );
⋮
  top=NULL; ch='a';
  while (ch!= 'E' )
  { 输入一个字符存入 ch;
    对 ch 进行判别并进行相应的处理;
    输出栈中的当前元素
  };
```

2．面向对象的方法

在面向对象的程序设计语言中，相关的数据及操作被统一在一个整体——对象之中。我们可以先将栈定义成类并创建一个栈对象，然后通过对该对象执行相应的操作来实现演

示程序的功能。以下是 Java 语言编写的链栈的类定义代码。

```
class Node{
  char data;
  Node next;
};
class LinkStack{
  private Node top;
  public LinkStack() {top=null;};
  public   void prnt() {…}
  public char pop() {…};
  public void push (char el){…}
};
```

由此创建一个链栈对象 lz1；然后通过对该对象执行相应的操作来实现演示程序的功能。例如，先将字符元素“a”、“b”推入栈中，然后再显示这个栈，可执行下述代码：

```
LinkStack lz1=new LinkStack();
lz1.push('a');lz1.push('b');lz1.prnt();
```

上述代码段不仅比较简洁，容易理解，而且也不会随类中实现细节的改变而改变。

面向对象实现方法充分体现了抽象数据类型的思想，可以将数据类型的 ADT 定义用一个类定义来表示。使用面向对象的实现方法来实现抽象数据类型比较贴近，也比较自然。

1.3.4　面向对象的概念

下面介绍面向对象实现方法中的有关概念。

1．对象

对象是指应用问题中所出现的各种实体，它是由一组属性值和在这组值上的一组操作(方法)构成的，其中，属性值确定了对象的状态。例如，在程序中常用的字符串、线性表、栈、队列等或在 Windows 应用程序中常见的窗体、组合框、编辑框、无线按钮等。对于一个编辑框对象，由它的 Top、Left 属性确定了它在窗体中的位置，由它的 Text 属性确定了显示在该编辑框中的字符串，可以使用 GetTextLen 方法来求得该字符串的长度，也可用 Clear 方法来清除这个字符串。

2．类和实例

对象在语言中是用类来定义的，类中定义了与某一种对象相关联的一组数据以及施与该数据中的一组基本操作，对象是数据与方法(操作)的统一体。在面向对象的程序设计中，如果某一个变量被定义成属于某一个类，那么该变量即可成为这种对象中的一个实例。因此，对象与实例这两个概念在程序设计中可对应于类与变量。对象、类是抽象的概念，而实例、变量代表具体事物。例如，在一个窗体中可以设置多个编辑框，尽管其位置及外观都不相同，但它们都是属于编辑框组件(类)所派生的对象实体。

3．数据封装(信息隐蔽)

数据封装是指在面向对象的程序设计中，对象的实现过程(包括数据的存储方式、操作