普通高等院校“十二五”规划教材

数据结构与算法实例教程

付学良　李宏慧　主　编
董改芳　亢汇涓　副主编

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

普通高等院校“十二五”规划教材

数据结构与算法实例教程

主　编　付学良　李宏慧

副主编　董改芳　亢汇涓

参　编　王艳芬　杨　婷　扈　华

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书是在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”课程的指导性大纲的指导下进行编写的。

本书结合内蒙古农业大学计算机与信息工程学院学生的实际情况，前半部分以案例驱动的方式引入每种数据结构，描述了它们的定义、抽象数据类型、存储结构和相关算法及应用；后半部分主要讨论查找和排序的各种实现方法及其综合分析比较。全书采用面向对象的C++作为数据结构和算法的描述语言，对于某种存储方式下的数据结构建立相应的类，类中成员函数就是对这种数据结构的操作，不涉及太多C++语言语法方面的知识，浅显易懂。

本书叙述清晰、语言简洁，注重基本知识的描述，既便于教学，又便于自学。

本书适合作为普通高等院校计算机相关专业本科、专科教材，也可以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。

图书在版编目（CIP）数据

数据结构与算法实例教程/付学良，李宏慧主编. —
北京：中国铁道出版社，2012.4
普通高等院校“十二五”规划教材
ISBN 978-7-113-14561-3

Ⅰ.①数… Ⅱ.①付… ②李… Ⅲ.①数据结构－高等学校－教材 ②算法分析－高等学校－教材 Ⅳ.①TP311.12

中国版本图书馆CIP数据核字（2012）第072632号

书　　名：数据结构与算法实例教程
作　　者：付学良　李宏慧　主编

策　　划：吴宏伟　孟　欣　　**读者热线：400-668-0820**
责任编辑：孟　欣　徐盼欣
封面设计：刘　颖
封面制作：白　雪
责任印制：李　佳

出版发行：中国铁道出版社（100054，北京市西城区右安门西街8号）
网　　址：http://www.51eds.com
印　　刷：三河市华业印装厂
版　　次：2012年4月第1版　　2012年4月第1次印刷
开　　本：787mm×1092mm　1/16　印张：17.25　字数：415千
印　　数：1～3 000册
书　　号：ISBN 978-7-113-14561-3
定　　价：33.00元

前言

FOREWORD

“数据结构”是一门研究计算机的操作对象、操作对象之间关系和操作对象的操作的学科，是介于数学、计算机硬件和计算机软件三者之间的一门核心课程，属于计算机学科中的一门综合性专业基础课程。它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和大型应用程序的重要基础。

本书主要介绍线性表、栈、队列、串、广义表和数组、树和二叉树、图等基本数据结构及其应用，以及查找和排序的原理与方法。通过本课程的学习，学生能较熟练地掌握数据结构的基本概念、特性、存储结构及相关算法；熟悉它们在计算机科学中最基本的应用；培养和训练运用高级程序设计语言编写结构清晰、可读性好的算法及初步评价算法的能力；为后续课程的学习，以及计算机软件的研制和开发打下一定的理论基础及实践基础。

本书具有以下特色：

1. 案例驱动。在介绍每一种数据结构前，采用案例引入的方式，确定该案例中用到的数据结构，分析案例中对这个数据结构需要进行哪些操作，怎样把现实中的例子转换成抽象的数据结构，并且实现这样的操作，给出大致的实现过程。案例取自现实生活，旨在一开始就引起学生的兴趣，避免在学习过程中理论与实际脱节。

2. “案例引入—数据结构—案例实现”层次清晰。书中第 1 章绪论比较特殊，第 9 章和第 10 章是两种不同操作，也比较特殊。除了这 3 章外，每一章都讨论一种数据结构，首先引入案例，通过对案例的分析引出相关数据结构的定义、术语、存储结构等知识，这些知识介绍完毕后，再讨论如何实现这个案例。这样组织章节内容，旨在既使学生学到理论知识，又使学生了解这种数据结构的应用实例。

3. C++描述。全书使用面向对象的 C++语言描述数据结构和算法，特点是不涉及太多 C++语言方面的知识，尽可能少地使用 C++的特性。实现每一种数据结构时，首先考虑这种结构用什么存储方式存储。只需创建一个类，在类中定义私有成员和公有函数。公有函数就是这种结构对应的各种操作。实现该结构时，只需定义一个类变量，然后访问这个类变量的成员函数即可。将重点放在如何分析每种算法的思想和复杂度上，了解 C++的读者只要了解了算法思想，也一定会灵活运用 C++语言实现这样的算法。

本书编者于 2009 年将“数据结构”建设为校级精品课程，在课程网站上提供了丰富的教学资源。2012 年成功申请内蒙古自治区精品课程。

本书适合作为普通高等院校计算机相关专业本科、专科教材，也可以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。

本书由付学良、李宏慧任主编，由董改芳、亢汇涓任副主编。其中第 8 章由付学良编写，第 5 章由李宏慧编写，第 1、10 章由亢汇涓编写，第 7 章由董改芳编写，第 2 章由杨婷编写，第 3、4 章由王艳芬编写，第 6、9 章由扈华编写。全书由付学良统稿。

由于编者知识水平有限，加上时间仓促，书中难免有不妥与疏漏之处，恳请专家和读者批评指正。

编者

2012 年 3 月

目录

CONTENTS

第 1 章　绪论 1
1.1　引言 1
1.2　数据结构的主要概念与术语 2
1.3　抽象数据类型的概念与描述 4
1.3.1　基本数据类型的概念 4
1.3.2　抽象数据类型 5
1.4　算法的度量 8
1.4.1　算法的定义 8
1.4.2　算法效率的度量 9
1.5　面向对象 C++描述工具简介 11
1.5.1　函数的定义格式 11
1.5.2　函数模板 11
1.5.3　类的定义 12
小结 16
习题 17
第 2 章　线性表 18
2.1　案例引入及分析 18
2.1.1　学生基本信息管理 18
2.1.2　线性表的定义 18
2.1.3　线性表的存储结构 20
2.2　学生基本信息管理之顺序表的实现 22
2.2.1　学生基本信息管理之顺序表类定义 22
2.2.2　学生基本信息管理之顺序表操作实现 23
2.2.3　学生基本信息管理之顺序表的主程序的实现 28
2.2.4　顺序表的其他操作 30
2.3　学生基本信息管理之单链表实现 31
2.3.1　学生基本信息管理之单链表类定义 32
2.3.2　学生基本信息管理之单链表操作实现 32
2.3.3　学生基本信息管理之单链表的主程序的实现 40
2.3.4　单链表的其他操作 42
2.4　算法分析 44
2.5　循环链表和双向链表 46
2.5.1　循环链表 46
2.5.2　双向链表 47
2.5.3　双向链表的类定义 47
2.6　静态链表 51

2.7 顺序结构与链表结构的比较 54
小结 54
习题 55
第 3 章 堆栈 58
3.1 案例引入及分析 58
3.1.1 提交批改作业 58
3.1.2 堆栈的定义 58
3.1.3 堆栈的存储结构 59
3.2 提交批改作业的顺序实现 64
3.3 提交批改作业的链式实现 65
3.4 算法分析 67
3.5 堆栈的其他应用 67
3.5.1 堆栈与递归的实现 67
3.5.2 表达式求值 69
3.5.3 背包问题 72
小结 75
习题 75
第 4 章 队列 77
4.1 案例的引入及分析 77
4.1.1 看病排队候诊 77
4.1.2 队列的定义 77
4.1.3 队列的存储结构 78
4.2 看病排队候诊的顺序实现 83
4.3 看病排队候诊的链式实现 85
4.4 算法分析 87
4.5 队列的其他应用 87
4.5.1 二进制数转换为十进制数 87
4.5.2 十进制数转换为二进制数 88
小结 89
习题 90
第 5 章 串 91
5.1 案例引入及分析 91
5.1.1 大整数计算器 91
5.1.2 串的定义 91
5.1.3 串的存储结构 93
5.2 大整数计算器的顺序实现 97
5.3 大整数计算器的链式实现 99
5.4 算法分析 101

5.5 串的其他应用 101
5.5.1 简单模式匹配 101
5.5.2 KMP 模式匹配 102
小结 106
习题 106
第 6 章 广义表和数组 108
6.1 案例引入及分析 108
6.1.1 本科生导师制问题 108
6.1.2 广义表的定义 108
6.1.3 广义表的存储结构 110
6.2 本科生导师制问题的实现 111
6.2.1 实现内容 111
6.2.2 实现过程 112
6.3 数组 120
6.3.1 数组的定义 120
6.3.2 数组的存储结构 122
6.4 矩阵的压缩存储 123
6.4.1 特殊矩阵的压缩存储 123
6.4.2 稀疏矩阵的压缩存储 124
小结 136
习题 136
第 7 章 树和二叉树 139
7.1 案例引入及分析 139
7.1.1 家谱管理 139
7.1.2 树和二叉树的定义 140
7.1.3 树和二叉树的存储结构 144
7.1.4 树与二叉树的转换 149
7.1.5 森林与二叉树的转换 151
7.1.6 树与森林的遍历 151
7.2 家谱管理的实现 152
7.3 遍历二叉树 154
7.3.1 前序遍历 154
7.3.2 中序遍历 155
7.3.3 后序遍历 156
7.3.4 按层次遍历 156
7.4 线索二叉树 157
7.5 树的其他应用——哈夫曼树及编码 161

7.5.1 哈夫曼树 161
7.5.2 哈夫曼编码 162
小结 163
习题 164
第 8 章 图 **165**
8.1 图的基本概念与术语 165
8.1.1 图的基本概念 165
8.1.2 图的基本术语 166
8.1.3 抽象数据类型 169
8.2 图的存储结构 171
8.2.1 邻接矩阵 171
8.2.2 邻接表 173
8.2.3 双链式存储结构 174
8.3 图的 ADT 设计与实现 179
8.4 图的遍历 180
8.4.1 深度优先搜索 181
8.4.2 广度优先搜索 183
8.5 图的连通性 184
8.5.1 无向图的连通分量和生成树 184
8.5.2 有向图的强连通分量 185
8.5.3 最小生成树 186
8.6 最短路径 192
8.6.1 单源最短路径 192
8.6.2 任意顶点间的最短路径 196
8.7 有向无环图及其应用 196
8.7.1 拓扑排序 196
8.7.2 关键路径 199
小结 202
习题 202
第 9 章 查找 **207**
9.1 查找的基本概念 207
9.2 静态查找表 208
9.2.1 顺序查找表 209
9.2.2 有序表的查找 209
9.2.3 静态索引顺序表的查找 212
9.3 动态查找表 213
9.3.1 二叉排序树和平衡二叉树 214

9.3.2 B-树和 B+树 226
9.4 哈希表 233
9.4.1 哈希表与哈希函数 233
9.4.2 哈希函数的构造方法 234
9.4.3 解决冲突的方法 236
9.4.4 哈希表的查找及其效率分析 239
小结 241
习题 242
第 10 章 排序 245
10.1 排序的基本概念 245
10.2 插入排序 246
10.2.1 直接插入排序 246
10.2.2 折半插入排序 247
10.2.3 2--路插入排序 249
10.2.4 表插入排序 250
10.2.5 希尔排序 251
10.3 交换排序 252
10.3.1 冒泡排序 252
10.3.2 快速排序 253
10.4 选择排序 256
10.4.1 简单选择排序 256
10.4.2 堆排序 257
10.5 二路归并排序 259
10.6 基数排序 260
10.6.1 多关键字排序 260
10.6.2 链式基数排序 260
10.6.3 各种排序方法的比较 265
小结 266
习题 266

第1章 绪　论

随着信息时代的到来，现实世界中的信息量急剧膨胀，很多实际应用问题使用手工方法已无法完成信息的管理，应运而生的计算机正好适合用于存储、处理数据量大、数据种类多的信息，而且处理速度快，可长久保存。在大量的信息中，通常有两种主要数据：一种是数值型数据，如整型、实型等；另一种是非数值型数据，如字符串、表、图形、图像、声音等。其中，非数值型数据占信息总量的一大部分，这些大量的非数值型数据是如何在计算机中存储及处理的呢？在计算机中进行存储及处理时又遵循什么规律、具备什么特点呢？这类问题，正是“数据结构”课程要介绍的主要内容。

1.1 引　言

利用计算机解决实际问题一般是将问题的解决步骤设计成算法，将算法设计成可执行程序，然后运行该程序得出结果。对于数值型问题，通常可以找到对应实际问题的数学公式，然后针对数学公式编写程序；而对于大多数的非数值型问题，如企业合同管理问题、职工信息管理问题、学生成绩管理问题等，处理的数据量很大，大部分是一些表格，如合同信息表、职工信息表、学生成绩表等，这些问题主要是对这些表格进行操作，如插入、删除、排序、查找、更新等，用单一的数学公式已经实现不了这些非数值型问题中的操作，对于这些大量的非数值型数据，在计算机中存储时不能是随意地堆放在存储器中，而应依据数据之间满足的关系特点将它们有规律地存储起来，然后设计出高效率的算法，这也是数据结构主要研究的问题。依据数据之间的关系特点，可以将数据结构分成4类，分别是：线性结构、树结构、图结构和集合。

【例 1-1】学生成绩管理问题。

在学生信息管理系统中，学生成绩管理通常包含学号、姓名、班级、高等数学、外语、计算机、体育等信息，根据每个学生的相关信息，可建立学生成绩表，如表1-1所示。

表 1-1　学生成绩表

	学　号	姓　名	班　级	高等数学	外　语	计算机	体　育
a_1	011012	赵明明	011005	98	89	80	70
a_2	011017	王乐	011009	87	90	77	85
a_3	011020	周小君	011004	80	88	90	75
a_4	011032	孙天乐	011005	90	73	60	65
…	…	…	…	…	…	…	…

在该表中，每一行作为一个完整的学生信息。当有新同学转入该校时，在该表中可完成新

学生信息的插入；有某同学转出时，要删除该学生的信息；还可以按给出的查找条件进行查询，或根据需要进行排序。在表 1-1 中，每一行之间逻辑上存在一种简单的线性关系。

【例 1-2】域名树形空间。

在网络中，计算机之间的正确通信是基于 IP 地址实现的，每台计算机都配置有一个 IP 地址。为了解决 IP 地址难以记忆的问题，各计算机可以使用域名来进行通信，如 www.imau.edu.cn；DNS 服务器负责解析出域名对应的 IP 地址，整个 DNS 域名空间被划分成许多个区域，这些域呈现树形结构，如图 1-1 所示。

【例 1-3】各个城市构成的通信网问题。

假设在 m 个城市之间要建立通信网，任何两个城市之间都有通信线路相连，如图 1-2 所示。

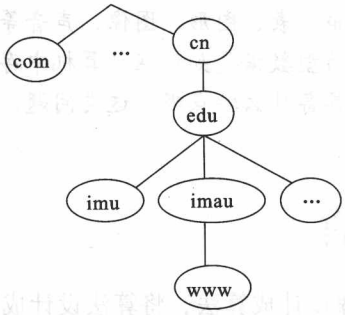


图 1-1　DNS 域名空间的树形结构

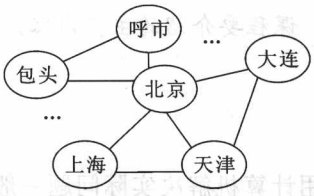


图 1-2　m 个城镇之间的通信网

在图 1-2 中顶点表示城市，顶点之间的连线表示两城市之间的通信线路。在计算机中描述这类问题时，需要描述清楚顶点以及顶点间的连线关系。这种关系不同于前面的线性结构和树结构，而是更复杂的结构，属于数据结构中的图。

由此可见，描述这类非数值问题的数学模型不再是数学方程，而是诸如表、树和图之类的数据结构。因此，简单说来，数据结构是一门研究非数值计算的程序设计问题中计算机操作对象以及它们之间的关系和操作等的学科。

“数据结构”于 1968 年由西方国家引入，在开始阶段包括图、表、树、集合代数、关系等内容，是计算机专业的一门专业基础课程，综合性、理论性、抽象性、复杂性均很突出，因此需要配合适量的实践才能很好地掌握。“数据结构”的研究不仅涉及计算机硬件的研究范围，特别是存储设备、编码、存取方法等，而且与计算机软件的关系也十分密切，在编译理论、操作系统等计算机专业课程中都有所应用。在“数据结构”课程不断发展、不断完善的历程中，抽象数据类型的观点是研究数据结构的一个重要切入点，采用目前广泛流行的面向对象程序设计语言来进行描述，在解决实际问题时可以恰当地反映现实事物的本质特征。

1.2　数据结构的主要概念与术语

本节将对一些基本概念、名词术语给出明确的定义，这些概念和术语在本书各章节中统一使用。

数据（data）：数据是对客观事物的符号表示，在计算机领域指所有能输入到计算机中被计算机程序处理的符号的总称。随着计算机处理能力的增强，图像、声音等多媒体数据都可以在计算机中进行加工处理，所以现在数据的含义已经非常广泛，不再局限于单一的数值类型。

数据元素（data element）：数据元素是数据的基本单位，在计算机程序中作为一个整体进行考虑和处理。如例 1-1 中表格的一横行就是一个数据元素，在表 1-1 中每个数据元素包括学号、姓名、班级、高等数学、外语、计算机、体育 7 个数据项（data item），数据项是数据的不可分割的最小单位。图 1-1 所示树形结构中的每一个域，如 cn 域、edu 域，以及图 1-2 中的每一个顶点都是一个数据元素。

数据对象（data object）：数据对象是性质相同的数据元素的集合，是数据的一个子集合。例如，字母数据对象是集合{'A','B','C',…,'Z'}，每个数据元素都是字符；表 1-1 所示学生成绩表也是一个数据对象，每个数据元素都具有学号、姓名、班级、高等数学、外语、计算机、体育 7 个数据项的值；整数数据对象包括{0, ±1, ±2, ±3,…}，每个数据元素都是整数。

数据结构（data structure）：数据结构是相互之间存在一种或多种关系的数据元素的集合。下面从集合的概念出发说明什么是关系。要定义关系，需要先定义什么是笛卡儿积。

笛卡儿积和关系：假设已知两个集合 A、B，A 和 B 的笛卡儿积表示为 $A\times B$，为下面有序偶对的集合：$A\times B=\{<x,y>|x\in A,y\in B\}$。将 $A\times B$ 的每一个子集都称为在 $A\times B$（或在 A 上，当 $A=B$ 时）上的一个关系。

设 r 是集合 M 上的一个关系，如果有 $<a,b>\in r$，则称 a 是 b 的直接前驱，b 是 a 的直接后继。

数据结构的形式：数据结构是一个二元组 Data_Structure=(D,S)，其中 D 是数据元素的有限集合，S 是 D 上关系的有限集合。

【例 1-4】已知数据结构二元组 $B=(K,R)$，其中：数据元素集合 $K=\{k_1,k_2,k_3,k_4,k_5\}$，关系集合 $R=\{<k_1,k_2>,<k_2,k_3>,<k_3,k_4>,<k_4,k_5>\}$，如图 1-3 所示。

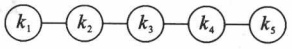
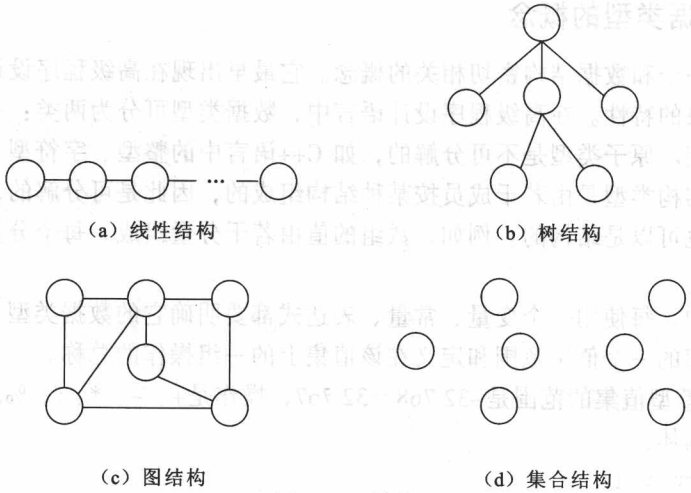


图 1-3 例 1-4 图

可以看出，该数据结构中数据元素之间呈现线性关系。

结构（structure）：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，将结构分为 4 类，如图 1-4 所示。



（a）线性结构　（b）树结构

（c）图结构　（d）集合结构

图 1-4 结构

（1）线性结构：数据元素之间存在一对一的关系。

（2）树结构：数据元素之间存在一对多的关系。

（3）图结构（或网状结构）：数据元素之间存在多对多的关系。

（4）集合结构：数据元素堆放在一个集合中，没有其他关系。

数据结构研究的是带结构的数据元素，上述4种结构描述的是数据元素之间的逻辑关系。讨论数据结构的目的是要在计算机中实现数据结构上的操作，因此还需研究数据结构在计算机中的存储表示方法。

数据结构在计算机中的存储表示称为数据的物理结构，也称为存储结构。既然数据结构中的数据元素之间是具有某种关系的，那么在存储数据结构时，一方面要存储数据元素本身，另一方面要表示关系。数据元素之间的关系在计算机中有两种不同的表示方法，因此存在两种不同的存储结构：顺序存储结构和链式存储结构。

顺序存储结构的特点是依据数据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，并采用一组地址连续的存储单元依次存储各个数据元素。例1-4线性结构中的元素 k_1、k_2、k_3、k_4、k_5采用顺序存储结构，如图1-5所示。

链式存储结构的特点是通过存储元素地址来表示数据元素之间的逻辑关系，并且每个数据元素在存储器中的存储地址可以是不连续的。例1-4线性结构中的元素 k_1、k_2、k_3、k_4、k_5采用链式存储结构，如图1-6所示。

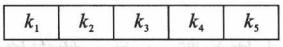


图1-5 顺序存储结构

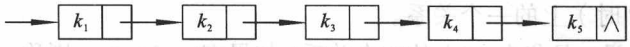


图1-6 链式存储结构

数据的逻辑结构与存储结构密切相关，在算法设计过程中都有很重要的作用。

1.3 抽象数据类型的概念与描述

要想深入理解抽象数据类型的概念，必须先掌握基本数据类型的概念及其使用方法。下面介绍高级语言中使用的基本数据类型。

1.3.1 基本数据类型的概念

数据类型是一个和数据结构密切相关的概念。它最早出现在高级程序设计语言中，用于描述程序中操作对象的特性。在高级程序设计语言中，数据类型可分为两类：一类是原子类型，另一类是结构类型。原子类型是不可分解的，如C++语言中的整型、字符型、浮点型、双精度型等基本类型。结构类型是由若干成员按某种结构组成的，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。

在高级语言中，每使用一个变量、常量、表达式都要明确它的数据类型，每一种数据类型都规定了可以取值的一个值集范围和定义在该值集上的一组操作的总称。

例如，16位整型值集的范围是-32 768～32 767，操作是+、-、*、/、%。又如，定义日期类型为下面的结构体：

```
Typedef struct {
    int  year;                    //年号
    int  month;                   //月号
    int  day;                     //日号
} DateType;                       //日期类型
```

该结构体类型定义后，就可以用来声明变量。在某种意义上，数据结构可以看成是“一组具有相同特性、相同结构的值集”，然后在数据结构上再定义一组操作的集合，如常见的插入操

作、删除操作等，这样就形成一种数据类型，这种类型不是高级语言中具有的，需要用抽象数据类型的方式来加以描述和定义。

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。抽象数据类型和数据类型概念实质是相同的，但抽象数据类型的范畴更广，常用于用户在设计软件系统时自定义的数据类型。数据结构可看作具有某种逻辑关系的一组值集，然后在此基础上定义一组操作，每一种数据结构当成一个抽象数据类型来定义，并且可以用三元组（D,S,P）来表示，其中 D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作集。

抽象数据类型定义格式为：

```
ADT 抽象数据类型名 {
    数据结构的定义{ 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{ 操作名称<参数表>
              初始条件
              操作结果
}ADT 抽象数据类型名
```

对于抽象数据类型的实现，本书采用面向对象的程序设计方法来实现抽象数据类型。在这种方法中，将抽象数据类型封装在一个类中，数据结构定义为类的数据部分，操作定义为类的方法部分。下面举例说明抽象数据类型的两种实现方法。

【例 1-5】抽象数据类型四元组的 ADT 方式的定义。

```
ADT  Four {
   数据对象：D={v1,v2,v3,v4∈ElemSet}(假设 ElemSet 为 int 类型)
   数据关系：R={<v1,v2>,<v2,v3>,<v3,v4>}
   基本操作:
   InitFour(&S,k1,k2,k3,k4)
   操作结果：构造一个四元组 S，元素 v1,v2,v3,v4 分别被赋予参数 k1,k2,k3,k4 的值。
   DestroyFour(&S)
   操作结果：销毁四元组 S。
   Get(S,i,&e)
   初始条件：四元组 S 存在，1≤i≤4。
   操作结果：用参数 e 返回 S 的第 i 个元素的值。
   Put(&S,i,e)
   初始条件：四元组 S 存在，1≤i≤4。
   操作结果：将参数 e 作为 S 的第 i 个元素的值。
   Max(S,&e)
   初始条件：四元组 S 存在。
   操作结果：用参数 e 返回 S 的 4 个元素中的最大值。
   Min(S,&e)
   初始条件：四元组 S 存在。
   操作结果：用参数 e 返回 S 的 4 个元素中的最小值。
}ADT Four
```

以下是抽象数据类型四元组的面向过程的实现。

```
#include <iostream.h>
#include <stdlib.h>
```

```
#define OVERFLOW  -2
#define OK  1
#define ERROR  0
typedef  int  Status;
typedef  int  * Four;
Status  InitFour(Four  &s,int  k1,int  k2,int  k3,int  k4){
   //构造四元组 s，依次给 s 赋予 4 个元素的值为 k1，k2，k3，k4
   s=new int[4];                          //分配 4 个元素的存储空间
   if(!s)  exit(OVERFLOW);                //分配存储空间失败
   s[0]=k1;s[1]=k2;s[2]=k3;s[3]=k4;
   return  OK;
}
Status  DestroyFour(Four &s){             //销毁四元组 s
   delete s;s=NULL;
   return OK;
}
Status  Get(Four  s,int  i,int  &e){      //1≤i≤4，用 e 返回 s 的第 i 个元素的值
   if(i<1||i>4)  return  ERROR;
   e=s[i-1];                              //第 i 个元素的下标为 i-1
   return  OK;
}
Status  Put(Four  &s,int  i,int  e){      //1≤i≤4，设置 s 的第 i 个元素的值为 e
   if(i<1||i>4)  return  ERROR;
   s[i-1]=e;                              //第 i 个元素的下标为 i-1
   return  OK;
}
Status  Max(Four  s,int  &e){        //用 e 返回四元组 s 的最大值
   e=s[0];
   if(s[1]>e)  e=s[1];
   if(s[2]>e)  e=s[2];
   if(s[3]>e)  e=s[3];
   return  OK;
}
Status  Min(Four  s,int  &e){        //用 e 返回四元组 s 的最小值
   e=s[0];
   if(s[1]<e)  e=s[1];
   if(s[2]<e)  e=s[2];
   if(s[3]<e)  e=s[3];
   return  OK;
}
   void main(){
   int a1,a2,a3,a4,m,n,s1,s2;
   Four  T;
   cin>>a1>>a2>>a3>>a4;
   InitFour(T,a1,a2,a3,a4);
   Get(T,3,m);
   cout<<"T 的第 3 个元素的值为: "<<m<<endl;
```

```
    cin>>n;
    Put(T,2,n);
    Max(T,s1);
    Min(T,s2);
    cout<<"T 的最大值为: "<<s1<<endl;
    cout<<"T 的最小值为: "<<s2<<endl;
    DestroyFour(T);
}
```

程序采用顺序存储结构存储四元组的各个元素的值，然后在程序中采用先定义、后调用的原则，先定义出抽象数据类型 Four 中的各个函数（即各个操作），如 InitFour()、DestroyFour()、Get()、Put()、Max()、Min()等函数，在主函数中，给实在参数 a1,a2,a3,a4 赋值以后，对自定义函数加以调用。请读者观察该程序的运行结果。

【例 1-6】 抽象数据类型“复数”的定义。

```
ADT Complex {
   数据对象: D={x1,x2 | x1,x2∈RealSet}
   数据关系: R1={<x1,x2>|x1是复数的实数部分，x2是复数的虚数部分}
   基本操作:
   AssignComplex(v1,v2)
   操作结果: 构造复数，其实部和虚部分别被赋以参数 v1 和 v2 的值。
   GetReal()
   初始条件: 某复数已存在。
   操作结果: 返回复数的实部值。
   GetImag()
   初始条件: 某复数已存在。
   操作结果: 返回复数的虚部值。
   Add(a1,b1)
   初始条件: 已知某复数存在。
   操作结果: 将参数 a1 与已知某复数的实部相加，将参数 b1 与已知某复数的虚部相加。
} ADT Complex
```

抽象数据类型“复数”面向对象的实现。

```
#include <iostream.h>
class  Complex{
private:
    float  a;
    float  b;
public:
    void  AssignComplex(float v1,float v2);
    float  GetReal(){return a;}
    float  GetImag(){return b;}
    void  Add(float a1,float b1);
};
void Complex::AssignComplex(float v1,float v2){
    a=v1;
    b=v2;
}
void  Complex:: Add(float a1,float b1){
    a+=a1;
    b+=b1;
```

```
}
void main(){
    Complex c1,c2;
    c1.AssignComplex(3.2,5.6);
    c2.AssignComplex(4.1,7.6);
    c1.Add(3.0,3.0);
    c2.Add(1.1,4.4);
    cout<<c1.GetReal()<<c1.GetImag()<<endl;
    cout<<c2.GetReal()<<c2.GetImag()<<endl;
}
```

在该程序中，首先定义了一个“复数”类，该类的数据部分就代表复数的实部与虚部，然后定义了类的成员函数，在主函数中分别调用了所定义的复数类的各个成员函数。请读者观察该程序的运行结果。

1.4 算法的度量

为了解决实际问题而设计的算法常常存在时间效率的高低与算法所占存储空间的大小两个方面的因素，两者往往不可兼得。下面介绍衡量算法效率的方法与概念。

1.4.1 算法的定义

算法（algorithm）是为解决某特定问题所采取的方法和步骤，是指令的有限序列，其中每一条指令表示一个或多个操作。

算法应该具有下列特性：

（1）**有穷性**：一个算法必须在有穷步之后结束，即必须在有限时间内完成。

（2）**确定性**：算法的每一步必须有确切的定义，无二义性。算法的执行对于相同的输入仅有相同的结果，在任何条件下仅有唯一的可执行流程。

（3）**可行性**：算法中定义的操作都是可以通过已经实现的基本运算的有限次执行得以实现的。

（4）**输入**：一个算法具有零个或多个输入，这些输入取自特定的数据对象集合。

（5）**输出**：一个算法具有一个或多个输出，这些输出同输入之间存在某种特定的关系。

在数据结构中，算法是用来描述操作步骤的方法，主要体现运算的设计思路、设计方法。为解决某一特定问题可以设计出不同的算法，一个好的算法通常要达到以下目标：

（1）**正确性**（correctness）：算法的执行结果应当满足具体问题的需求，对于实际问题应事先给出功能和性能的要求，至少应指出需要什么样的输入、输出，需要进行什么样的处理或计算。

（2）**可读性**（readability）：一个算法应当思路清晰、层次分明、简单明了、易读易懂，便于大家交流，晦涩难懂的算法容易隐藏错误。

（3）**健壮性**（robustness）：当输入不合法数据时，应能做出适当处理，不至于引起严重后果。

（4）**高效性**（efficiently）：算法应尽量占用较少的存储空间和较少的运行时间，以提高算法的空间与时间效率。