



普通高等教育“十二五”规划教材

数据结构 (C语言版)

(第二版)

袁和金 刘 军 牛为华 王翠茹 李宗民 编著



中国电力出版社
CHINA ELECTRIC POWER PRESS



普通高等教育“十二五”规划教



数据结构 (C语言版)

(第二版)

编著 袁和金 刘 军 牛为华 王翠茹 李宗民

主审 王熙照



中国电力出版社
CHINA ELECTRIC POWER PRESS

内 容 提 要

本书为普通高等教育“十二五”规划教材。本书内容涵盖了常用的各种数据结构，详细讨论了它们的逻辑特点、存储表示和实现原理，给出了算法实现的类C语言描述和复杂性分析。书中算法简捷，没有过多地涉及与编程语言和环境相关的细节，便于教师教学，也有助于学生能力的锻炼和培养。书中各章后面还配备了内容丰富的习题，并编写了配套的习题解答，便于学生练习和巩固所学的知识。另外，教材最后一章以动态内存管理、工程项目统筹图的建立和分析及游戏程序开发等为例展示了数据结构的相关知识在实际开发中的应用。

本书可作为高等本科院校数据结构课程的教材，也可作为高职高专相关课程的教材，还可供自学人员及工程技术人员学习参考。

图书在版编目（CIP）数据

数据结构：C语言版 / 袁和金等编著. —2版. —北京：中国电力出版社，2012.6
普通高等教育“十二五”规划教材
ISBN 978-7-5123-3247-8

I. ①数… II. ①袁… 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. ①TP311.12 ②TP312

中国版本图书馆CIP数据核字（2012）第145586号

中国电力出版社出版、发行

（北京市东城区北京站西街19号 100005 <http://www.cepp.sgcc.com.cn>）

航远印刷有限公司印刷

各地新华书店经售

*

2006年9月第一版

2012年9月第二版 2012年9月北京第二次印刷

787毫米×1092毫米 16开本 21.25印张 519千字

定价 39.00元

敬告读者

本书封底贴有防伪标签，刮开涂层可查询真伪
本书如有印装质量问题，我社发行部负责退换

版权专有 翻印必究



前 言

本书从抽象数据类型的观点出发,系统全面地介绍了数据结构课程中的基本理论、方法及技巧。全书共分概述、顺序表、链表、数组和广义表、字符串、树、图、查找表、内排序、外存储设备、外排序及数据结构的应用示例和一个附录。书中介绍了各种数据结构的定义和性质,详细分析和讨论了这些结构的逻辑特点、存储表示及在这些结构上定义的一些运算的实现方法及其复杂性分析。在每章的末尾配备了足够的习题,附录对实习步骤和内容做了较详细的介绍。

本书是作者在多年来讲授《数据结构》、《算法设计与分析》等课程的基础上,结合自身的教学体会和学生的建议,对2006年出版的C语言版教材进行了部分修订和补充而成。在本版中,补充了线索树的构造及基于线索树的二叉树遍历、图的十字链表的建立、基于Kruskal算法和Prim算法构造最小生成树、求关键路径、分块查找、基于链地址法的散列表查找、插入、删除和自然的两路归并等算法的类C语言描述,并在各章后面补充了内容丰富的习题,方便学生练习和巩固所需的知识。本书在内容上力求做到图文并茂、深入浅出,并对每个算法都阐述了其设计思想和实现方法,给出了其复杂性分析。

本书讲授60~90学时,内容可视学时的多少进行取舍,在学时较少的情况下,可以舍去如判定树、文件、外排序等内容。全书由袁和金和王翠茹统稿,河北大学数学与计算机学院王熙照教授认真审阅了全部书稿并提出了许多宝贵的修改意见,编者表示深切的谢意。

本书适于作为计算机科学与技术、软件工程、网络工程、信息安全及电子、信息相关专业的教材,也可供从事相关工作的科技与工程人员参考。

由于作者的水平和编写时间限制,书中必定存在一些错误和不足之处,恳切希望读者批评指正。

编 者
2012年5月

目 录

前言

第1章 概述	1
1.1 引言	1
1.2 问题的求解过程	2
1.3 基本概念	4
1.4 数据结构课程内容	7
1.5 类C语言和算法分析	8
习题一	12
第2章 顺序表	17
2.1 线性表	17
2.2 栈和队列	23
习题二	41
第3章 链表	46
3.1 单链表	46
3.2 链栈和链队	56
3.3 循环链表与多重链表	64
习题三	67
第4章 数组和广义表	73
4.1 数组的逻辑结构	73
4.2 广义表	86
习题四	89
第5章 字符串	93
5.1 字符串及其运算	93
5.2 字符串的存储表示	94
习题五	103
第6章 树	106
6.1 基本术语及性质	106
6.2 树的抽象数据类型和树的存储	108
6.3 二叉树	112
6.4 二叉树的遍历	119
6.5 二叉线索树	124
6.6 树的遍历	130
6.7 树的应用	132

习题六	140
第 7 章 图	146
7.1 基本术语	147
7.2 图的存储结构	149
7.3 图的遍历和求图的连通分量	157
7.4 生成树和最小生成树	161
7.5 最短路径	166
7.6 拓扑排序	172
7.7 关键路径	177
习题七	183
第 8 章 查找表	188
8.1 查找表的基本概念	188
8.2 静态查找表的实现	189
8.3 动态查找表的实现	197
8.4 Hash (杂凑) 法	222
习题八	230
第 9 章 内排序	235
9.1 计数排序	235
9.2 直接插入排序	237
9.3 折半插入排序	239
9.4 冒泡排序	240
9.5 希尔排序	241
9.6 快速排序	244
9.7 简单选择排序	246
9.8 堆排序	247
9.9 归并排序	252
9.10 基数排序	257
9.11 总结	262
习题九	262
第 10 章 外存储设备	269
10.1 外存储器	269
10.2 基本概念	271
10.3 顺序文件	273
10.4 索引文件	274
10.5 ISAM 文件	276
10.6 VSAM 文件	278
10.7 直接存取文件	280
10.8 链接式文件和多重表文件	281
10.9 倒排文件	282

第 11 章 外排序	284
11.1 外排序的主要过程	284
11.2 K 路归并	285
11.3 缓冲区的并行操作处理	287
11.4 初始归并段的产生	289
11.5 磁带归并排序	292
第 12 章 应用示例	299
12.1 多重链表的一个实际应用：动态存储管理	299
12.2 工程项目中统筹图的建立和分析	305
12.3 数据结构在游戏开发中的简单应用	310
附录 上机实验	320
实验一 线性结构的顺序表示	321
实验二 链表	322
实验三 数组和广义表	324
实验四 串	325
实验五 树	326
实验六 图	327
实验七 二叉排序树	328
实验八 排序	329
实验九 综合实验	330
参考文献	332

第 1 章 概 述

程序设计的实质是数据表示和数据处理,而这种表示和处理应通过一个渐进的过程逐步完成。数据结构课程主要讨论这个过程中一些最基本的问题。本章将概括地介绍有关数据结构的基本概念、基本思想及本课程所用的算法语言,最后给出了度量算法代价的大“O”表示法。

1.1 引 言

计算机科学是一门研究数据表示和数据处理的科学,数据则是计算机化的信息,它是计算机可以直接处理的最基本、最重要的对象,无论是进行科学计算或数据处理、过程控制及对文件的存储和检索乃至数据库技术等计算机应用领域中,都是对数据进行加工处理的过程。因此要设计出一个结构好、效率高的程序,必须研究数据的特性、数据间的相互关系及其对应的存储表示,并利用这些特性和关系设计出相应的算法和程序。这一过程的归纳和抽象就是数据结构这门学科产生和发展的由来。

数据结构作为一门独立的课程在国外是从 1968 年才开始的,但在此之前其有关内容已散见于编译原理及操作系统之中了。20 世纪 60 年代中期,美国的一些大学开始设立有关课程,但当时的课程名称并不叫“数据结构”,而被称作“表处理语言”,它的主要内容是研究当时已经出现的几种表处理语言。如 J.Weizenbaum 在 20 世纪 50 年代初设计的 SLIP 语言(简单表处理语言),50 年代后期由 A.Newell 等设计的 IPLV 语言(信息处理语言),1959 到 1960 年 J.McCarthy 设计的著名的 LISP 系统(表处理语言),以及 1962 由 D.Farber 等人设计的 SNOBOL 语言(串处理语言)。这些语言系统的共同特点是,处理的数据对象的结构形式或者是表结构,或者是树结构。如 LISP 语言的数据结构就是二叉树形式,SONBOL 的数据结构则是表和树的形式。

上述几种语言是以数据为中心,为处理非数值问题而设计的,如 LISP 就是为实现人工智能而设计的表处理语言。而像 Fortran、ALGOL 等一般人较为熟悉的算法语言则主要是为解决数值计算问题而设计的,它们侧重以建立程序为中心,只有当数据成为程序的加工对象时,才考虑到数据。这种观点显然适用于数值计算问题,这类问题属于在简单数据结构上进行复杂的函数变换问题。以数据为中心的观点则把数据结构作为问题的中心(如数据库),而把程序看成是围绕着数据结构缓慢爬行的小虫,它时而询问,时而修改或扩充当前放在内存中的数据。这种观点适用于诸如人事档案管理系统、航空订票系统及情报检索系统等非数值计算问题的处理,这些大的系统都要求采用复杂的数据结构描述系统的状态,它们的运算(或操作)是实现对数据结构的访问或修改等。

由于数据必须在计算机中进行处理,因此,除去对数据本身的数学特征需要加以注意和研究之外,还必须考虑数据的物理结构,以及这种数据结构的相关操作及其合理实现。

1.2 问题的求解过程

用计算机解决一个具体问题时，一般需要下列几个步骤：

- (1) 分析阶段：弄清所要解决的问题是什么，并且把它用某种方式清楚地描述出来；
- (2) 设计阶段：算法的设计、模块的设计和数据结构的设计，即建立程序系统的结构；
- (3) 编码阶段：根据系统结构的要求编写出可执行的程序；
- (4) 测试阶段：发现和排除在前几个阶段中产生的错误；
- (5) 维护阶段：对程序系统进行运行和管理。

数据结构课程并不研究由上述五个阶段构成的程序开发全过程，而是集中讨论以其中的设计阶段为核心，同时涉及编码和分析阶段的一个小范围内的若干基本问题。

例如，用回溯法解决组合锁问题。某种车的一种组合锁由 N 个开关组成，每个开关可以“闭合”或“打开”。假设当 $N/2$ 或更多一些开关处在“闭合”位置时，就可能打开这个锁。假设已经忘记了这个组合，而必须把这个锁打开，就得去试一试所有的组合，用回溯法来找到这个组合。

弄清问题是解决问题的前提，可用一个 N 位二进制数模拟每一种可能的组合。如果开关 i 闭合，则第 i 位为 1；如果开关 i 打开，则第 i 位为 0。利用一棵二叉树作为 N 个元素的不同状态的模型，这棵树第 K 级上的每一个结点，将对应于 N 位数中前 K 位的不同位置值。这一级上的每个结点又向下分成两个结点，这两个结点是相应这个 N 位数在第 $(K+1)$ 位上两个可能的位置值。本例仅考虑 $N=4$ 时的二叉树结构，如图 1.1 所示。

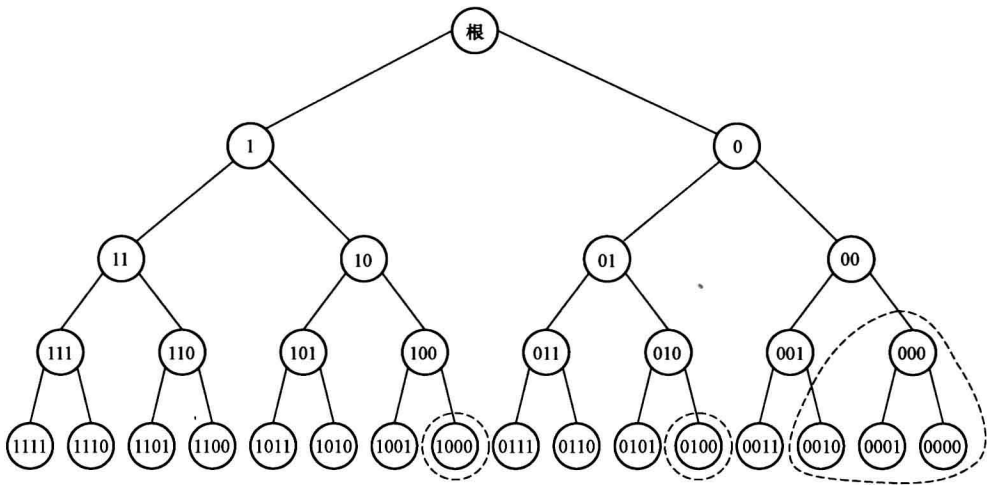


图 1.1 组合锁问题求解树模型

由于至少必须有 $N/2$ 个开关“闭合”才有可能满足打开这个条件，所以对于一些不能提供正确组合的分支便不去生成它。具体来说，如果结点含有两个 0，它的右支就不必去生成。图 1.1 中用虚线框框起部分，就不必再去生成。

下面介绍用二叉树作为组合锁问题的数学模型，生成至少有 $N/2$ 个开关“闭合”的那些组合的回溯过程。实际上是对这棵树的遍历。首先，从根结点出发，选取这棵树的最左一支，

即沿左分支向下直到终端结点，校验这个结点是否符合正确组合的条件，若不符合，则回溯一级；查找是否有没查过的分支，若有，则沿着没查过的分支向下搜索，搜索到终端结点时校验，若不符合打开条件（正确组合条件）便回溯；若所有分支都查过便直接回溯一级，再按同样的方法搜索。对图 1.1 用回溯法按下面的次序访问结点。

```

根
  1
    11
      111
        1111 判断这个组合，若不是，则回溯，否则结束；
          111  向下取右分支；
        1110 判断这个组合，若不是，则回溯；
          111  左右分支已查完，再回溯；
            11  取右分支；
          110  取左分支；
        1101 判断这个组合，若不是，则回溯；
          110  取右分支；
        1100 判断这个组合，若不是，则回溯；
          110  无待查结点，回溯；
            11  无待查结点，回溯；
              1  取右分支；
            10  取左分支；
          101  取左分支；
        1011 判断这个组合，若不是，则回溯；
          101  取右分支；
        1010 判断这个组合，若不是，则回溯；
          101  无待查结点，回溯；
            10  取右分支；
          100  取左分支；
        1001 判断这个组合，若不是，则回溯；
          100  因为 100 的右分支不满足至少有两个 1 的限制，因此不搜索 100 的右分支，而回
溯到 10；
            10  无待查结点，回溯；
              1  无待查结点；回溯；
            根  取右分支；
            ⋮

```

当再次返回到根结点时，或已经找到了正确的组合时，算法结束。

从上面例子可见，描述这类非数值计算问题的数学模型不再是数学方程，对上例采用树这种数据结构来描述，如要解决数据库管理方面的问题则可用表结构来描述，如要解决旅游花费问题则可用图结构来描述。

用计算机求解问题的一般过程, 可用如图 1.2 所示的框图来概括。

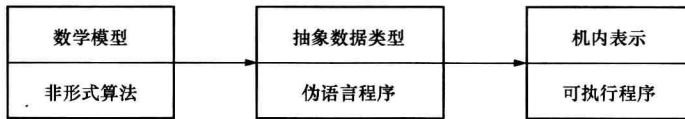


图 1.2 问题求解过程

从图 1.2 可以看出, 要想解决一个实际问题, 首先要建立恰当的数学模型来描述要处理的问题, 并设计出一个求解该问题的初步的“非形式算法”。然后, 用逐步求精的方法对“非形式算法”细化, 用伪语言写出比较形式的算法, 对那些在算法中用到的非基本的数据类型建立抽象数据类型, 用过程名给这个类型上的每个操作命名, 并且用对这些过程的调用来取代算法中的每个操作。最后, 对每个抽象数据类型选择一种实现的方法, 同时编写出这些抽象数据类型上定义的所有操作的实现过程。若伪语言程序中还有不可执行的语句, 要加以细化, 从而得到全部是可执行的程序。这一过程的每一步骤都包括数据表示与数据处理两个方面。在上述过程中, 数据表示的形式经历了如下变化: 机外表示→数学模型→抽象数据类型→机内表示。数据处理方面的变化过程是: 处理要求→非形式算法→伪语言程序→可执行程序。

1.3 基本概念

本节介绍本书中常用的名词和术语的含义。

1. 数据 (data)

随着计算机科学的发展和计算机应用的普及, 计算机加工处理的对象已从早期的数值、布尔值等扩展到字符串、表格、图像、声音等。因此, 凡是能被计算机存储、加工的对象通称为数据, 它是计算机程序加工的“原料”。

2. 数据元素和数据项 (data element and item)

数据元素是数据的基本单位, 在程序中作为一个整体加以考虑和处理。有时, 一个数据元素可由若干个数据项组成, 数据项是数据的不可分割的最小单位。

3. 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。例如, 整数数据对象是集合 $D = \{0, \pm 1, \pm 2, \dots\}$, 字母字符数据对象是集合 $C = \{ 'A', 'B', \dots, 'Z' \}$ 。

4. 数据结构 (data structure)

数据结构不同于数据类型, 也不同于数据对象, 它不仅涉及数据类型和数据对象, 而且要描述数据对象各元素之间的相互关系。这种数据元素之间的相互关系称为结构。根据数据元素之间关系的不同, 通常有如图 1.3 所示的四类基本的结构形式。

- (1) 集合: 集合中的任何两个结点之间都没有逻辑关系, 组织形式松散;
- (2) 线性结构: 结构中元素之间存在一对一的关系, 依次排列形成一条“锁链”;
- (3) 树形结构: 结构中元素之间存在一对多的关系, 具有分支、层次特性;
- (4) 图状结构: 结构中元素间存在多对多的关系, 元素间互相缠绕, 任何两个元素都可

以邻接。

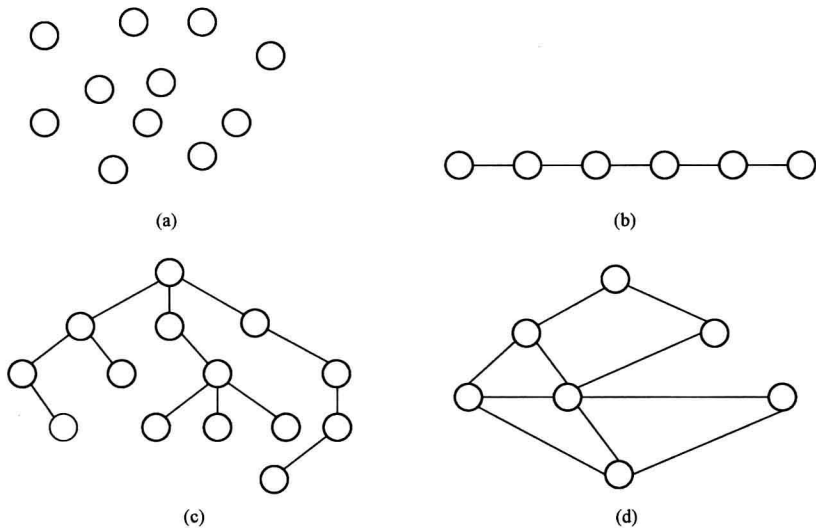


图 1.3 四种基本结构关系图

(a) 集合；(b) 线性结构；(c) 树形结构；(d) 图状结构

数据结构可以用二元组的形式来进行数学意义上的形式定义： $\text{Data_structure}=(D, S)$ 。

其中： D 是数据元素的有限集， S 是 D 上关系的有限集。它是从操作对象中抽象出来的数学模型，仅仅描述了数据元素之间的某种逻辑关系。为了在计算机中实现对数据元素的操作，还需要考虑数据关系在计算机中的存储。因此数据结构应包括三方面的内容：

- (1) 数据的逻辑结构：数据元素之间的逻辑关系；
- (2) 数据的存储结构：数据元素及其关系在计算机存储器中的表示；
- (3) 数据的运算：对数据对象施加的操作。

(1) 数据的逻辑结构 (logic structure of data)。从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，数据的逻辑结构可以看作是从具体问题上抽象出来的数学模型，通常把数据元素之间的关联方式（邻接关系）称为数据元素间的逻辑关系。数据元素之间逻辑关系的整体称为逻辑结构。

(2) 数据的存储结构 (storage structure of data)。数据结构在计算机中的表示称为数据的存储结构，它包括数据元素的表示和关系的表示。在计算机中表示信息的最小单位是二进制数的一位，叫做位 (bit)。在计算机中，可以用一个若干位组合起来形成的一个位串表示一个数据元素，通常称这个位串为元素或结点 (node)。当数据元素由若干数据项组成时，位串中对应于各个数据项的子位串称为数据域 (data field)。元素间的关系在计算机内的表示方法通常有四种。

1) 顺序存储方法：该方法是指每个存储结点只含有一个数据元素，所有存储结点相继存放在一个连续的存储区里。用存储结点间的位置关系表示数据元素之间的逻辑关系。

2) 链式存储方法：该方法是指每个存储结点不仅含有一个数据元素，还包含一组指针。每个指针指向一个与本结点有逻辑关系的结点。

3) 索引存储方法：该方法是指在存储结点信息的同时，还建立索引表，索引表的每一项

称为索引项，索引项的一般形式为：(关键字，地址)。关键字是能唯一标识一个结点的那些数据项，地址表示该关键字所在结点的起始存储位置。

4) 散列存储方法：该方法是根据结点的关键字值，采用某种方法，直接计算出该结点的存储地址。

5. 数据类型 (data type)

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。如 C 语言中的整数类型，其值集为 $[-\text{Max int}, \text{Max int}]$ 上的整数，定义在其上的一组操作为：加、减、乘、整除和取模等。按“值”的不同特性，数据类型可分为非结构的原子类型和结构类型两类。原子类型的值是不可分解的，如 C 语言中的标准类型（整型、实型、字符型、布尔型）、枚举类型、子界类型和指针类型。结构类型的值是由若干成分按某种结构组成的，因此其值是可分解的，并且它的成分可以是非结构的，也可以是结构的。例如，一个记录类型的元素的值由若干个分量组成，每个分量可以是整数或数组，也可以是指针类型等。引入“数据类型”的目的，仅从使用数据类型的角度来说，实现了信息的隐蔽，将一切不必了解的细节都封装在类型中。例如，用户在使用“整数”类型时，既不需要了解“整数”在机内的表示，也不需要知道其操作是如何实现的。如“两整数求和”，程序设计者注重的仅仅是其“数学上求和”的抽象特性，而不是其硬件的“位”操作如何进行。

6. 抽象数据类型 (abstract data type)

抽象数据类型和数据类型实质上是一个概念，是指一个数学模型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，而与其在机内的表示和实现无关，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。抽象数据类型与数据类型相比范畴更广些，它不再局限于前述程序语言中已定义并实现的数据类型（固有的数据类型），还包括用户在设计软件系统时自己定义的数据类型。

假设要设计一个圆 (Circle) 为抽象数据类型，它提供的操作除了按给定半径构造一个圆外，还要包括计算面积 (area) 和周长 (circumference)。下面是 Circle 的一个抽象数据类型定义，关于这个类型的具体实现（包括数据结构的表示和完成操作的代码）请读者自己完成：

```
ADT circle is
  data
    非负实数表示圆的半径
  operations
    constructor
      输入的初值：非负实数
      处理：构造一个圆
    area
      输入：无
      输出：圆的面积
      处理：计算圆的面积
    circumference
      输入：无
      输出：圆的周长
      处理：计算圆的周长
end ADT circle
```

和数据结构的形式定义相对应，抽象数据类型可以用以下三元组表示：

$$\text{ADT} = (\text{D}, \text{S}, \text{P})$$

式中： D 是数据对象，用结点的有限集合表示； S 是 D 上的关系集，用结点间序偶的集合表示； P 是对 D 的基本操作集。其操作的定义格式为：

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>

【例 1.1】 构造复数，参考代码如下：

ADT Complex

{

数据对象： $D = \{e_1, e_2 \mid e_1, e_2 \in \text{Realset}\}$

数据关系： $R_1 = \{ \langle e_1, e_2 \rangle \mid e_1 \text{ 是复数的实数部分, } e_2 \text{ 是复数的虚数部分} \}$

基本操作：

```

InitComplex(e1, e2);           //初始化：以 e1 和 e2 为实部和虚部构造一个复数
DestroyComplex(Z);            //销毁复数 Z
GetReal(Z, realPart);         //得到复数 Z 的实部
GetImag(Z, ImagPart);        //得到复数 Z 的虚部
Add(Z1, Z2, Sum);             //复数 Z1 和 Z2 相加，结果保存到 Sum 中
Subtract(Z1, Z2, Difference); //复数 Z1 和 Z2 相减，结果保存到 Difference 中
Multiply(Z1, Z2, Product);    //复数 Z1 和 Z2 相乘，结果保存到 Product 中

```

}ADT Complex

至于上述抽象类型如何具体实现，读者可以根据所学过的知识来思考，此处不再具体介绍。

7. 基本运算 (basic operation)

基本运算只描述处理功能，不包括处理步骤和方法，是在逻辑结构上的操作。

8. 运算实现 (realization of operation)

运算实现的核心是处理步骤的规定，即算法设计。对一个复杂的运算仍需按照逐步求精的方法构造它的实现。

9. 算法 (algorithm)

算法是指解决某一特定类型问题的有限运算序列。一个算法应该具有下列特性：

- (1) 有穷性：一个算法必须总是在执行有穷步之后结束；
- (2) 确定性：算法的每一步，必须有确切的定义，读者理解时不会产生二义性；
- (3) 输入：一个算法有零个或多个输入，这些输入取自于特定的对象集合；
- (4) 输出：一个算法有一个或多个输出，它们是同输入有某种特定关系的量；
- (5) 可行性：算法应该是能行的，即算法中所有待实现的运算都是能够正确地进行的。

1.4 数据结构课程内容

数据结构与数学、计算机硬件和软件有着十分密切的关系。数据结构是介于数学、计算机硬件和计算机软件之间的一门计算机科学专业的核心课程，是高级程序设计语言、编译原理、操作系统、数据库、人工智能等课程的基础。同时，数据结构的技术也广泛应用于信息科学、系统工程、应用数学，以及各种工程技术领域。

数据结构课程集中讨论软件开发过程中的设计阶段、同时涉及编码和分析阶段的若干基

本问题。此外，为了构造出好的数据结构及其实现，还需考虑数据结构及其实现的评价与选择。因此，数据结构课程的内容包括三个层次的五个“要素”，其内容体系如表 1.1 所示。

表 1.1 数据结构课程内容体系

层 次 \ 方 面	数据表示	数据处理
抽象	逻辑结构	基本运算
实现	存储结构	算法
评价	不同结构的比较及算法分析	

1.5 类 C 语言和算法分析

1.5.1 类 C 语言

本书算法的描述采用类 C 语言，它精选了 C 语言的一个核心子集，同时做了若干扩充和修改，增强了语言的描述功能。以下对其进行简要说明。

(1) 数据结构的表示（存储类型）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行定义。

(2) 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
```

除了函数的参数需要说明类型外，算法中使用的辅助变量可以不做变量说明，必要时对其作用给予注释。一般而言，`a`、`b`、`c`、`d`、`e` 等用作数据元素名，`i`、`j`、`k`、`l`、`m`、`n` 等用作整型变量名，`p`、`q`、`r` 等用作指针变量名。

为了便于描述，在函数表中除了值调用方式外，增添了 C++ 语言的引用调用的参数传递方式。在形参表中，以 `&` 打头的参数即为引用参数。引用参数能被函数本身更新参数值，可以此作为输出数据的管道。参数表中的某个参数允许预先用表达式的形式赋值，作为默认值使用，以简化参数表。

(3) 内存的动态分配与释放过程为：

- 1) 使用 `new` 和 `delete` 动态分配和释放内存空间；
- 2) 分配空间 指针变量=`new` 数据类型；
- 3) 释放空间 `delete` 指针变量。

(4) 赋值语句有：

```
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 2, ..., 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 \leftrightarrow 变量名;
 条件赋值 变量名=条件表达式? 表达式 T: 表达式 F;

(5) 选择语句有:

条件语句 1 if(表达式)
 {
 语句序列;
 }

条件语句 2 if(表达式)
 {
 语句序列;
 }
 else
 {
 语句序列;
 }

开关语句 1 switch(表达式)
 {
 case 值 1: {语句序列 1; break;}
 case 值 2: {语句序列 2; 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 2: {语句序列 2; break;}
 ...
 case 条件 n: {语句序列 n; break;}
 default: {语句序列 n+1;}
 }

(6) 循环语句有:

for 语句 for(赋值表达式序列; 条件; 修改表达式序列)
 {
 语句序列;
 }

while 语句 while(条件)
 {
 语句序列;
 }

do-while 语句 do
 {
 语句序列;
 } while(条件)

(7) 结束语句有:

函数结束语句 return 表达式;
 return;

异常结束语句 `exit` (异常代码);

(8) 输入和输出语句有:

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);

(9) 注释:

单行注释 //文字序列;

(10) 基本函数有:

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);

求进位整数 `ceil` (表达式);

求不足整数 `floor` (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`;

(11) 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。

1.5.2 算法的评价

解决同一问题, 常常可以设计出不同的算法, 如何比较这些算法的优劣是一个有意义的问题。如本书第八章对排序问题就介绍了多种不同的方法, 这些方法的排序速度和占用内存的多少存在很大的差异。评价算法的目的, 既是为了从解决同一问题的不同算法中选出较为适用的一种, 同时也有助于考虑对现有的算法如何进行改进或设计出新的算法。

1. 评价算法的一般原则

一个好的算法通常应主要具有如下特点。

(1) 正确性: 指算法在允许的输入数据范围内, 能在有穷时间内得出正确的结果。关于算法正确性的证明需要使用有关的数学理论 (如集合论、代数学的定理及离散数学等知识), 严格地说它属于程序设计方法学课程所研究的内容。通常, 对于较复杂的问题, 可以将算法分解成一些局部的模块来分析, 只有每个模块都是正确的, 才能保证整个算法的正确性, 对于本书中所讨论的算法, 有些正确性是显而易见的, 有些则对其正确性做了简单证明, 其他的则将正确性证明过程省略了, 因为这部分内容不属于本书所要讨论的范围。

在实践中, 对于算法的正确性, 一般采用测试的方法来验证。算法的正确性可以分为以下四个层次: 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、苛刻而带有刁难性的几组输入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产生满足规格说明要求的结果。显然, 达到第④层意义下的正确是极为困难的, 所有不同输入数据的数量相当大, 逐一验证的方法是不现实的。对于大型软件需要进行专业测试, 而一般情况下, 通常以第③层意义的正确性作为衡量一个程序是否合格的标准。

(2) 算法执行速度快: 即依据算法编程后在计算机上运行时所消耗的时间越少越好。

(3) 算法所占用的空间较省: 即依据算法编程后在计算机上运行时所占的存储量较少。