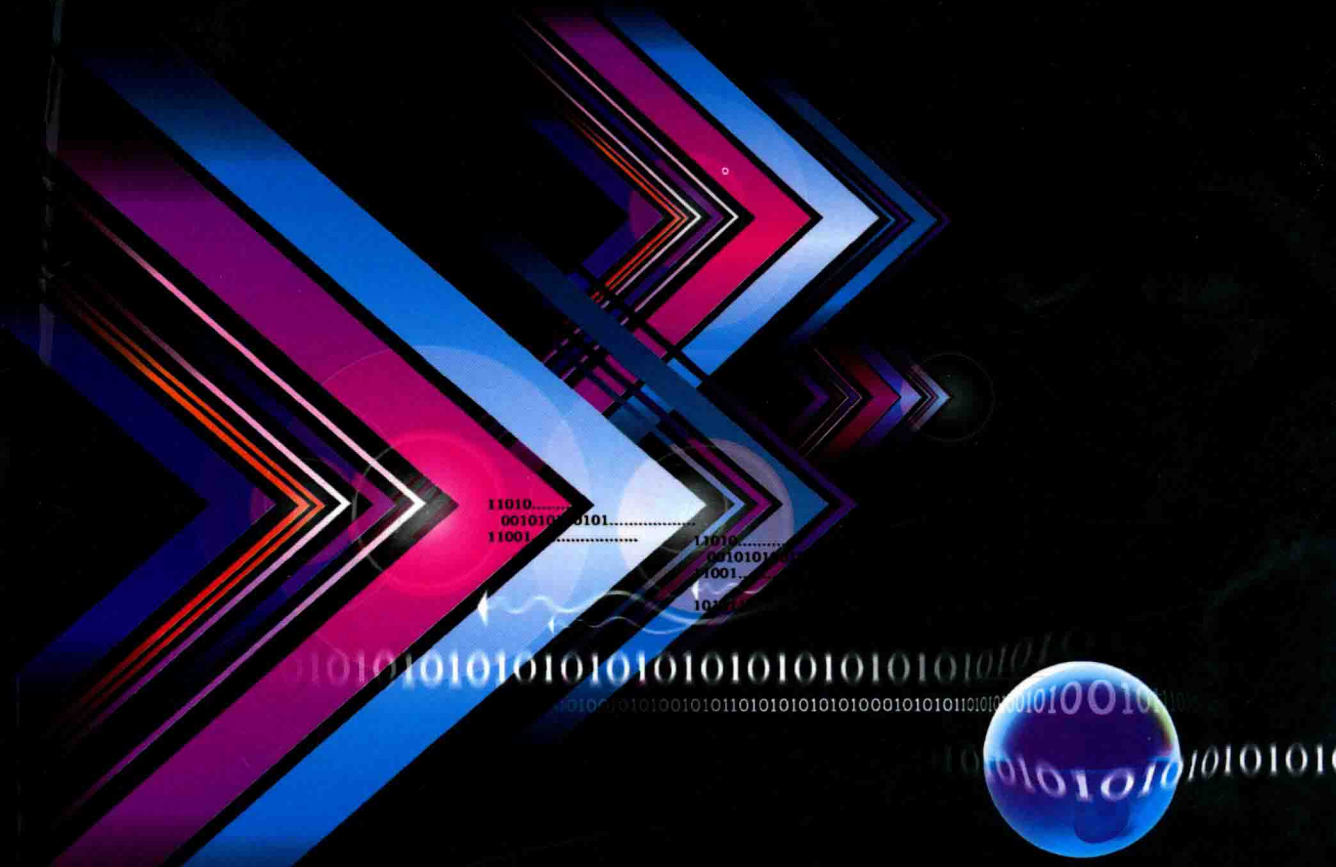
中国电子教育学会高教分会推荐

普通高等教育电子信息类“十三五”课改规划教材



数据结构实用教程

（C语言版）

主编　王欣欣　冷玉池

西安电子科技大学出版社
http://www.xduph.com

中国电子教育学会高教分会推荐

普通高等教育电子信息类“十三五”课改规划教材

数据结构实用教程

(C 语言版)

主　编　王欣欣　冷玉池

副主编　刘　伟　李炳荣　胡自强

西安电子科技大学出版社

内 容 简 介

本书系统地介绍了各种常用的数据结构与算法方面的基本知识，并使用 C 语言描述其算法。详细介绍了如何用 C 语言表示具体的数据结构，及其在具体 C 程序中的应用，从而使学生在深刻理解数据结构的基础上，灵活运用 C 语言知识解决实际问题。

全书共 8 章，第 1 章介绍了数据结构与算法的一些基本概念；第 2～6 章分别讨论了线性表、栈与队列、串、树和二叉树、图等常用的数据结构及其应用；第 7 章和第 8 章分别介绍了查找和内部排序，它们都是广泛使用的数据处理技术。全书配有大量的例题和详尽的注释，各章都有不同类型的习题和实验，并配有可执行的 C 程序代码。

本书可作为普通高等教育应用型本科院校理工科学生的教材，也可作为高职高专和成人教育的教材，还可作为高等学校计算机专业硕士研究生入学考试的复习用书，对从事计算机系统软件、应用软件的设计与开发的人员及计算机编程爱好者也有很好的参考作用。

图书在版编目(CIP)数据

数据结构实用教程：C 语言版/王欣欣，冷玉池主编. —西安：西安电子科技大学出版社，2016.2
普通高等教育电子信息类“十三五”课改规划教材
ISBN 978-7-5606-4005-1

Ⅰ. ① 数… Ⅱ. ① 王… ② 冷… Ⅲ. ① 数据结构—高等学校—教材 ② C 语言—程序设计—高等学校—教材 Ⅳ. ① TP311.12 ② TP312

中国版本图书馆 CIP 数据核字(2016)第 020898 号

策　　划　毛红兵
责任编辑　刘玉芳　毛红兵
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
网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.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　2016 年 2 月第 1 版　2016 年 2 月第 1 次印刷
开　　本　787 毫米×1092 毫米　1/16　印　张　10
字　　数　231 千字
印　　数　1～3000 册
定　　价　21.00 元
ISBN 978-7-5606-4005-1/TP

XDUP 4297001-1

前　言

在科技高速发展的信息时代，计算机是处理信息的主要工具，计算机知识已成为人类当代文化的一个重要组成部分，它的应用已从传统的数值计算领域发展到各种非数值计算领域。在非数值计算领域里，数据处理的对象已从简单的数值发展到一般的符号，进而发展到具有一定结构的数据。然而面临的主要问题是：针对每一种新的应用领域的处理对象，如何选择合适的数据表示(结构)，如何有效地组织计算机存储，以及此基础上如何有效地实现对象之间的“运算”关系。传统的数值计算的许多理论、方法和技术已不能满足解决非数值计算问题的需要，必须进行新的探索。数据结构就是研究和解决这些问题的重要基础理论。因此，“数据结构”课程已成为计算机类专业一门重要的专业基础课。它涉及在计算机中如何有效地表示数据、合理地组织和处理数据，还涉及初步的算法设计和算法性能分析技术。学好数据结构课程，将为后续的专业课程，如数据库系统、操作系统、编译原理等打下良好的知识基础，而且还为软件开发和程序设计提供了必要的技能训练。

本书按照教育部高等学校计算机基础课教学指导分委员会的《数据结构基本教学要求》编写。本书结合作者多年的实践教学经验，针对学生在利用数据结构编程解决实际问题时困难较大这一情况，更新教学内容和方法，以 C 语言为工具，系统介绍数据结构的知识与应用，因此，本课程要求学生对 C 语言编程、程序设计方法有一定基础。

本书内容由浅入深，简明扼要，通俗易懂，重点突出，语言简练流畅，算法全部采用 C 语言描述，很容易转换成程序，可读性好，应用性强，便于教与学，有利于学生理解和掌握数据结构中数据表示和数据处理的方法，适应新形势下普通高等院校学生的特点。本书以学生为主体，层次清晰，注重应用，习题丰富，有利于提高学生的思维能力和综合应用能力。

本书将数据结构理论和 C 语言编程有机地融合起来，提高学生解决实际问题的程序设计能力，激发学生的学习兴趣。全书配有大量的例题和详尽的注释，且各章都有不同类型的习题、实验，并配有可执行的 C 程序代码。

参与本书编写工作的教师都有十年以上教授数据结构课程的教学经历，教学经验丰富。他们不仅积累了多年的实践教学经验，还参与过大量的科研实践，使得他们对计算机前沿领域发展有正确的了解和把握，从而保证本书既能反映本领域基础性、普遍性的知识，保持内容的相对稳定性，又能紧随科技的发展及时调整、更新内容。

因编者水平有限，书中不当之处在所难免，恳请读者和同行批评指正，联系方式 caomuxinxin@163.com。

编　者

2015 年 10 月

目　录

第1章 绪 论

“数据结构”是计算机程序设计中一门重要的理论技术基础课程，它不仅是计算机学科的核心课程，而且也是其他理工科专业的热门选修课。“数据结构”不仅涉及计算机硬件(特别是编码理论、存储装置和存取方法等)的研究范围，而且和计算机软件的研究有密切的关系。“数据结构”是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。著名计算机科学家沃思教授提出的公式：程序 = 算法 + 数据结构，也说明了数据结构的重要性。

1.1 什么是数据结构

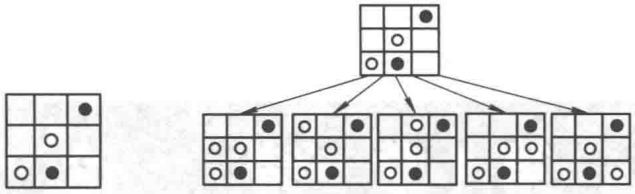
简单地说，数据结构是一门研究非数值计算的程序设计问题中，计算机的操作对象以及它们之间的关系和操作等的学科。

例 1.1 学生信息表。表 1.1 所示是一种数据结构，表中的每一行是一个记录(计算机操作的对象)，记录由学号、姓名、班级、性别和出生年月等数据项组成。在这种数据结构中，计算机的主要操作是按照某个特定要求(如给定学号)对信息表进行查询的，计算机处理的对象间存在一种线性关系，称为线性数据结构。

表 1.1 学 生 信 息 表

学号	姓名	班级	性别	出生年月
201457506116	王浩	光 126	男	1996.1
201457506117	杨帅	光 126	男	1996.10
201457506118	高波	光 126	男	1995.1
…	…	…	…	…

例 1.2 计算机和人对弈问题。计算机之所以能和人对弈是因为对弈的策略事先已存入计算机。由于对弈的过程是在一定规则下随机进行的，为使计算机能灵活对弈，就必须把对弈过程中所有可能发生的情况以及相应的对策都考虑周全。在对弈中，计算机操作的对象是对弈过程中可能出现的棋盘状态(称为格局)，格局之间的关系是由比赛规则决定的。通常，这个关系不是线性的，从一个棋盘格局可以派生出几个格局，例如从图 1.1(a)所示的棋盘格局可以派生出 5 个格局，如图 1.1(b)所示。若将从对弈开始到结束过程中所有可能出现的格局都画在一张图上，则可得到一棵倒长的“树”，称为树型数据结构。



(a) 棋盘格局示例　　(b) 对弈树的局部

图 1.1　井字棋对弈“树”

例 1.3　制订教学计划。在制订教学计划时，需要考虑各门课程的开设顺序。有些课程需要先导课程，有些课程则不需要，而有些课程又是其他课程的先导课程。如某高校计算机专业必修课程的开设情况如表 1.2 所示。

表 1.2　课　程　表

课程编号	课程名称	先修课
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1，C2
C4	汇编语言	C1
C5	语言设计和分析	C3，C4
C6	计算机原理	C11
C7	编译原理	C3，C5
C8	操作系统	C3，C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C1，C9，C10

可将这些课程的先后关系用图 1.2 所示的形式描述。

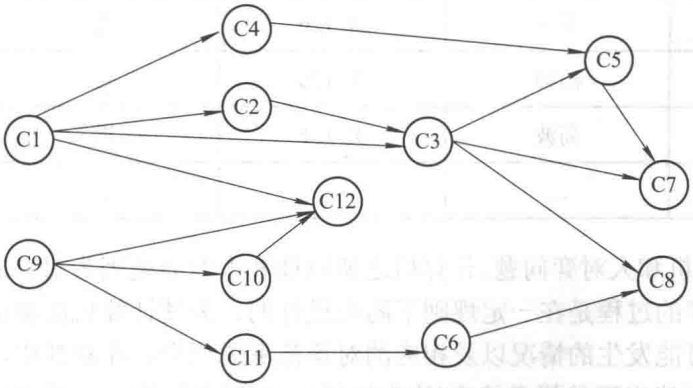


图 1.2　课程先后关系图

图 1.2 中，用圆圈代表课程，圆圈中的字符代表课程的编号，称为顶点。若某门课程在开设之前必须先修另一门课程，则在这两个顶点之间就有一条由先导课程指向该课程的有向弧，这种结构被称为图形结构。通过对图形结构的操作，就可以得到一种课程的线性

排列顺序，该顺序一定能够满足先导课程在后序课程前开设的要求。

综合上述 3 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、树和图之类的数据结构。

1.2 基本概念和术语

本节将对一些常用的概念和术语进行介绍，这些概念和术语在以后的章节中会多次出现。

1. 数据(Data)

数据是外部世界信息的载体，它能够被计算机识别、存储和加工处理，是计算机程序加工的原料。计算机程序处理各种各样的数据，可以是数值数据，如整数、实数或复数；也可以是非数值数据，如字符、文字、图形、图像、声音等。

2. 数据元素(Data Element)和数据项(Data Item)

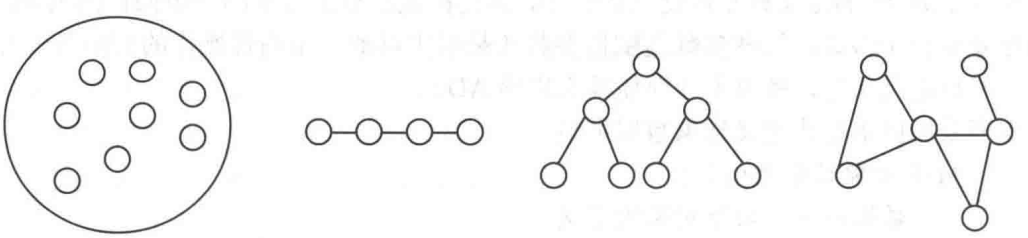
数据元素是数据的基本单位，在计算机程序中通常被作为一个整体进行考虑和处理。数据元素有时也被称为元素、结点、顶点、记录等。一个数据元素可由若干个数据项组成。数据项是不可分割的、含有独立意义的最小数据单位。例如，例 1.1 中学生信息表中的一条记录就是一个数据元素，这条记录中的学号、姓名、性别、班级、出生年月等字段就是数据项。

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 N = {±1，±2，…}，字母字符数据对象是集合 C = {'A', 'B'，'C'，…，'Z'}。

4. 数据结构(Data Structure)

数据结构是相互之间存在一种或多种特定关系的数据元素的集合。在任何问题中，数据元素之间都不是孤立的，而是存在着一定的关系，这种关系称为结构(Structure)。根据数据元素之间关系的不同特性，通常有四类基本数据结构：① 集合，如图 1.3(a)所示，该结构中的数据元素除了存在“同属于一个集合”的关系外，不存在任何其他关系。集合中元素的关系极为松散，可用其他数据结构来表示。② 线性结构，如图 1.3(b)所示，该结构中的数据元素存在着一对一的关系。③ 树型结构，如图 1.3(c)所示，该结构中的数据元素存在着一对多的关系；④ 图状结构或网状结构，如图 1.3(d)所示，该结构中的数据元素存在着多对多的关系。



(a) 集合　(b) 线性结构　(c) 树型结构　(d) 图状结构

图 1.3 四类基本数据结构关系图

数据结构的形式定义可用一个二元组 Data_Structure = (D，S)表示，其中，D 是数据元

素的有限集，S 是 D 上关系的有限集。下面举两个简单例子说明。

例 1.4　表 1.1 的数据结构。

StudentInfo = (D，S)

其中，D 是光 126 班级全体学生集合 D = {Stu_1，Stu_2，…，Stu_n | n 为班级人数}。S 是定义在集合 D 上的一种关系 S = {<Stu_i，Stu_{i+1}> | i=1，2，…，n}，有序偶<Stu_i，Stu_{i+1}>表示学号相邻。

例 1.5　表 1.2 的数据结构。

TeachPlan=(D，S)

其中，D 是计算机专业必修课程集合 D = {C1，C2，…，C12}。S 是定义在集合 D 上的一种关系 S = {<C1，C4>，<C1，C2>，<C1，C3>，<C1，C12>，…| 共 16 对}，有序偶表示课程的先导关系，例如<C1，C4>表示“程序设计基础”是“汇编语言”的先导课程。

上述数据结构定义中的“关系”描述的是数据元素之间的逻辑关系，因此又称为数据的逻辑结构。讨论数据结构的目的是为了在计算机中实现对它的操作，因此还需研究它如何在计算机中表示。

数据结构在计算机中的具体存储(又称映像)称为数据的物理结构，又称存储结构。它包括数据元素的表示和逻辑关系的表示。数据元素之间的关系在计算机中有两种不同的表示方法：顺序映像和非顺序映像，并由此得到两种不同的存储结构：顺序存储结构和链式存储结构。数据的逻辑结构和存储结构是密切相关的两个方面，任何一个算法的设计取决于数据的逻辑结构，而算法的实现依赖于采用的存储结构。

如何描述存储结构呢？由于本书是在高级程序语言的层次上讨论数据结构的操作的，因此，可以借用高级程序语言中提供的“数据类型”来描述存储结构。例如可以用所有高级程序语言中都有的“一维数组”类型来描述顺序存储结构，以 C 语言提供的“指针”来描述链式存储结构。

5．抽象数据类型(Abstract Data Type，ADT)

抽象数据类型是指一个数学模型及定义在该模型上的一组操作。它可以看做是数据的逻辑结构及在此结构上定义的一组操作。

ADT 只指出数据的逻辑结构及其操作，至于该逻辑结构以何种存储方式实现，以及这些操作如何实现等细节对用户来说是隐蔽的，并且通过封装来阻止外部对抽象数据类型直接访问。所以，ADT 实际上是数据结构作为一个软件组件的实现。

对一个 ADT 的定义就是约定它的名字，约定在该类型上定义的一组操作的名字，明确各操作要多少个参数，这些参数是赋值参数还是引用参数，指明各操作的初始条件与操作结果。一旦定义清楚，就可十分简便地引用该 ADT。

本书采用以下格式定义抽象数据类型：

```
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
}
```

其中，数据对象和数据关系的定义用伪码描述，基本操作用函数描述。基本操作有两种参数：赋值参数和引用参数，其中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。

一个含抽象数据类型的软件模块通常应包含定义、表示和实现三个部分。

例 1.6 抽象数据类型矩形的定义。

一个矩形必须具备长度和宽度，因此，矩形的抽象数据类型的基本数据项应该是矩形的长(Length)和宽(Width)。另外，对矩形的基本操作应该包括：构造矩形长度、求矩形的周长、求矩形的面积。

```
ADT Rectangle{
  数据对象：D={ Length , Width | ∈实数}
  数据关系：R={< Length , Width >}
  基本操作：
  InitRectangle(Rectangle &rec, h, w)
  操作结果：构造了一个矩形，矩形的长和宽分别被赋以参数 h，w 的值。
  GetPerimeter(Rectangle rec, &e)
  初始条件：矩形 rec 存在；
  操作结果：用 e 返回 rec 的周长。
  GetArea(Rectangle rec, &e)
  初始条件：矩形 rec 存在；
  操作结果：用 e 返回 rec 的面积。
}
```

抽象数据类型和数据类型实质是同一个概念。例如，整数类型是一个 ADT，其数据对象是指能容纳的整数，基本操作有加、减、乘、除和取模等。抽象数据类型比通常的数据类型范畴更广，比如用户可以把线性表、栈等更复杂的数据结构定义为抽象数据类型。由于抽象数据类型是数据结构作为软件组件的实现，能提高软件的复用率，所以抽象数据类型在操作系统、数据库系统等大型系统软件和应用软件的设计中得到了广泛的应用。

1.3 抽象数据类型的实现

抽象数据类型可通过固有数据类型来表示和实现，并用已经实现的操作来组合新的操作。由于本书是在高级程序设计语言的层次上讨论抽象数据类型的表示和实现、数据结构及其算法的实现及验证，前后涉及的代码比较多，放在一起比较繁琐，算法的重点也不突出，故采用分模块的方法，把需要的公共文件做成一个模块。现作简要说明。

1. 预定义常量、类型和所需要的头文件

```
#include<string.h>
#include<ctype.h>
#include<malloc.h>          // malloc()等
#include<stdio.h>           // EOF(=^Z 或 F6)，NULL
```

```
#include<math.h>          // floor(),ceil(),abs()
#include<process.h>       // exit()
#define TRUE 1            // 函数结果状态代码
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
typedef int Status;       // Status 是函数的类型，其值是函数结果状态代码，如 OK 等
typedef int Boolean;      // Boolean 是布尔类型，其值是 TRUE 或 FALSE
typedef int ElemType;     // 数据元素定义，用户在使用该数据类型时也可自行定义
```

本书把以上的代码作为一个独立的头文件，命名为 head1-1.h，以便被后续章节的程序使用。

2. 存储结构用类型定义(typedef)描述

例 1.7　抽象数据类型 Rectangle 的实现。

```
typedef struct{
      float Height;
      float Width;
}Rectangle;//矩形抽象数据类型的存储结构
void InitRectangle(Rectangle &rec, float h, float w)
{
    rec.Height=h;  //操作结果：初始化矩形，长为 h，宽为 w
    rec.Width=w;
}
void GetPrimeter(Rectangle rec, float &pre)
{                                  //初始条件：矩形存在
 pre=rec.Height*2+rec.Width*2;     //操作结果：pre 返回矩形的周长
}
void GetArea(Rectangle rec, float &area)     //初始条件：矩形存在
{
   area=rec.Height*rec.Width;                //操作结果：area 返回矩形的面积
}
#include "stdio.h"
 int main()                                  //用主程序检验 3 个基本操作是否完成其功能
{
   float e;
   Rectangle rect;
   InitRectangle(rect, 4.5, 5.5);            //构造一个高为 4.5 m，长为 5.5 m 的矩形
   GetPrimeter(rect, e);                     //用 e 返回该矩形的周长
```

```
    printf("该矩形周长为: %f\n", e);
    GetArea(rect, e);                    //用 e 返回该矩形的面积
    printf("该矩形面积为: %f\n", e);
    return 0;
}
```

1.4 算法和算法分析

算法与数据结构和程序的关系非常密切。进行程序设计时，先确定相应的数据结构，然后再根据数据结构和问题的需要设计相应的算法。由于篇幅所限，下面只从特性、要求和时间复杂度等三个方面对算法进行介绍。

1.4.1 算法的特性

算法(Algorithm)是对某一特定类型的问题求解步骤的一种描述，是指令的有限序列，其中每条指令表示一个或多个操作。一个算法应该具备以下 5 个特性：

(1) 有穷性：一个算法总是在执行有穷步之后结束，即算法的执行时间是有限的。

(2) 确定性：算法的每一个步骤都必须有确切的含义，即无二义，并且对于相同的输入只能有相同的输出。

(3) 输入：一个算法具有零个或多个输入。输入是在算法开始之前给出的量，这些输入是某数据结构中的数据对象。

(4) 输出：一个算法具有一个或多个输出，并且这些输出与输入之间存在着某种特定的关系。

(5) 可行性：算法中的每一步都可以通过已经实现的基本运算的有限次运行来实现。

算法的含义与程序非常相似，但二者有区别。一个程序不一定满足有穷性，例如操作系统，只要整个系统不遭破坏，它将永远不会停止；一个程序只能用计算机语言来描述，程序中的指令必须是机器可执行的，而算法不一定用计算机语言来描述，自然语言、框图、伪代码都可以描述算法。

本书将尽可能采用 C 语言来描述和实现算法，使读者能够阅读或上机执行，以便更好地理解算法。

1.4.2 算法设计的要求

通常设计一个“好”的算法应考虑达到以下目标：

(1) 正确性：算法的执行结果应当满足预先规定的功能和性能的要求，这是评价一个算法的最重要和最基本的标准。算法的正确性还包括对于输入、输出处理的明确而无歧义的描述。

(2) 可读性：算法主要是为了人阅读和交流，其次才是机器的执行。一个算法应当思路清晰、层次分明、简单明了、易读易懂。即使算法已转变成机器可执行的程序，也需要考虑人能较好地阅读和理解。同时，一个可读性强的算法也有助于排除算法中隐藏的错误

和移植算法。

(3) 健壮性：一个算法应该具有很强的容错能力，当输入不合法的数据时，算法应当能做适当的处理，不至于引起严重的后果。健壮性要求算法要全面细致地考虑所有可能出现的边界情况和异常情况，并对这些边界情况和异常情况做出妥善的处理，尽可能使算法没有意外的情况发生。

(4) 高效率与低存储量需求：一般而言，效率指的是算法执行的时间。对于同一个问题，如果有多个算法可以解决，则执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。效率与存储量需求两者都与问题的规模有关，好的算法应该在具有高效率的同时，存储量需求也较低。

1.4.3 算法的时间复杂度

一个算法的时间复杂度是指该算法的运行时间与问题规模的对应关系。一个算法是由控制结构和原操作构成的，其执行的时间取决于二者的综合效果。

为了便于比较同一问题的不同算法，通常把算法中基本操作重复执行的次数(频度)作为算法的时间复杂度。算法中的基本操作一般是指算法中最深层循环内的语句，因此，算法中基本操作语句的频度是问题规模 n 的某个函数 f(n)，记作 T(n) = O(f(n))。其中“O”表示随问题规模 n 的增大，算法执行时间的增长率和 f (n)的增长率相同，或者说，用“O”符号表示数量级的概念。例如 T(n) = 0.5n(n−1)，则 0.5n(n−1)的数量级与 n^2 相同，所以 $T(n) = O(n^2)$。

如果一个算法没有循环语句，则算法中基本操作的执行频度与问题规模 n 无关，记作 O(1)，也称为常数阶。如果算法只有一个一重循环，则算法的基本操作的执行频度与问题规模 n 呈线性增大关系，记作 O (n)，也叫线性阶。常用的还有平方阶 $O(n^2)$、立方阶 $O(n^3)$ 和对数阶 $O(\log_2 n)$等。

例 1.8　分析以下程序的时间复杂度。

```
x=n;                    //n>1
y=0;
while (y<x)
{
    y = y+1;                // 语句①
}
```

这是一个一重循环的程序，while 循环的循环次数为 n，所以，该程序段中语句①的频度是 n，则程序段的时间复杂度是 T(n) = O(n)。

例 1.9　分析以下程序的时间复杂度。

```
for (i=1; i<n; ++i )
{
  for (j=1, j<n; ++j)
      A[i][j]=i*j;          // 语句①
}
```

这是一个二重循环的程序，外层 for 循环的循环次数是 n，内层 for 循环的循环次数为

n，所以，该程序段中语句①的频度为 n^2，则程序段的时间复杂度为 $T(n) = O(n^2)$。

例 1.10 分析以下程序的时间复杂度。

```
x=n;
y=0;
while (x>=(y+1)*(y+1))
{
    y=y+1; // 语句①
}
```

这是一个一重循环的程序，while 循环的循环次数为 $n^{1/2}$，所以，该程序段中语句①的频度是 $n^{1/2}$，则程序段的时间复杂度是 $T(n) = O(n^{1/2})$。

例 1.11 分析以下程序的时间复杂度。

```
for(i=1; i<=n; i++)
  for(j=1; j<=n; j++)
  {  A[i][j]=0;
     for(k=1; k<=n; k++)
       A[i][j]+= A[i][k]* A[k][j];
  }
```

整个算法的执行时间与该基本操作(乘法)重复执行的次数 n^3 成正比，故程序的时间复杂度为 $T(n) = O(n^3)$。

习 题

1. 简述下列术语。

   数据元素 数据项 数据结构 数据类型 数据逻辑结构 数据存储结构 算法

2. 数据结构课程的主要目的是什么？
3. 分别画出线性结构、树型结构和图状结构的逻辑示意图。
4. 什么是算法的时间复杂度？怎样表示算法的时间复杂度？