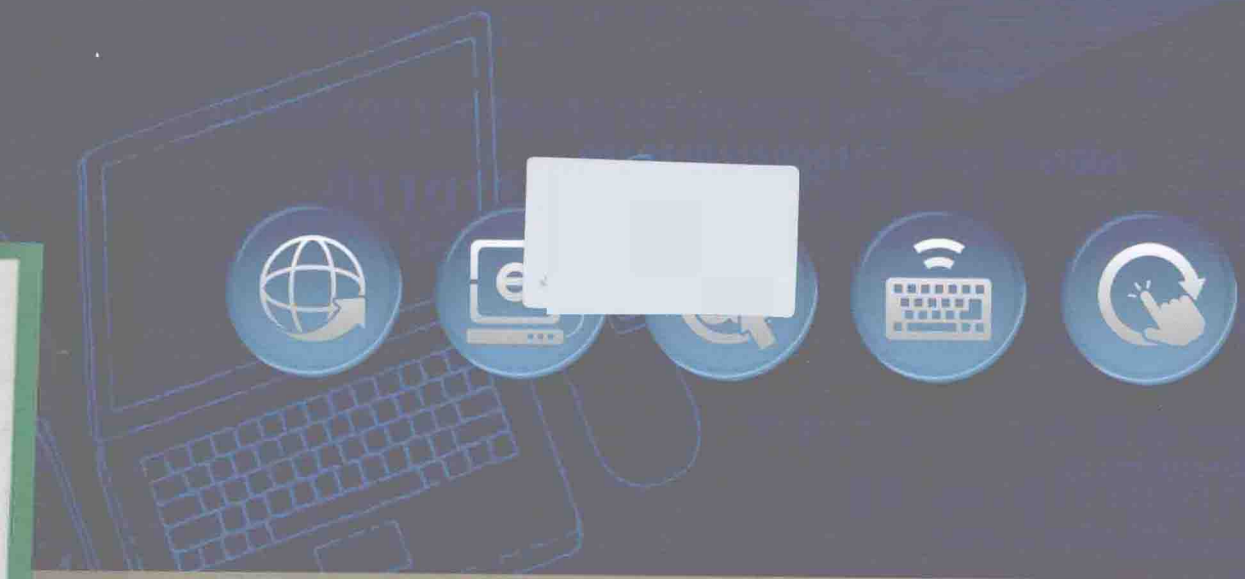
计 算 机 类 本 科 规 划 教 材

数据结构

——使用C语言（第5版）

◆ 朱战立 编著



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

计算机类本科规划教材

数据结构
——使用C语言
（第5版）

朱战立　编著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

数据结构是计算机学科各专业的一门重要的专业基础课。本书包含了2009年研究生入学统考大纲的全部内容。本书系统地介绍了线性表、堆栈、队列、串、数组、广义表、树、二叉树、图等典型数据结构，以及递归、查找和排序的方法。本书理论叙述简洁准确、实践应用举例丰富完整，从而达到理论和实践密切结合的教学目的。本书采用C语言描述算法。

本书内容丰富，难度适中，文字简洁准确，图文并茂，应用实例多，教学参考资料丰富。本书免费提供教学课件，可登录华信教育资源网（www.hxedu.com.cn）注册后下载。

本书既可作为计算机专业本科、专科学生的教材，也可供从事计算机工程和应用工作的科技工作者参考。

图书在版编目（CIP）数据

数据结构：使用C语言 / 朱战立编著. —5版. —北京：电子工业出版社，2014.1

计算机类本科规划教材

ISBN 978-7-121-21699-2

Ⅰ. ①数… Ⅱ. ①朱… Ⅲ. ①数据结构—高等学校—教材 Ⅳ. ①TP311.12

中国版本图书馆CIP数据核字（2013）第246563号

责任编辑：冉 哲
印 刷：北京市李史山胶印厂
装 订：北京市李史山胶印厂
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编 100036
开 本：787×1 092 1/16 印张：20 字数：537千字
印 次：2014年1月第1次印刷
印 数：4 000册 定价：40.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

数据结构是计算机学科各专业一门重要的专业基础课，也是其他计算机相关专业的一门必修课或选修课。数据结构课程的教学目的，是使学生掌握组织数据、存储数据以及处理数据的基本概念和软件设计的基本方法，从而为进一步学习后续专业课程打下坚实的基础。

本书作者20多年来一直从事数据结构课程的教学工作，曾编著过若干本采用不同算法描述语言的数据结构教材。本书是在经过长期使用的以前出版的教材基础上，参照新的研究生入学统考大纲，通过作者进一步修改、补充和完善而成的。

2009年出版的本教材第4版，包含了2009年研究生入学统考大纲的全部内容。经过近5年的使用，作者发现原书内容稍嫌过多，像“文件”一章的内容，大多数学校已不再讲授。本次修订出版的第5版，删除了“文件”一章，以及原第1章中算法书写规范的内容。对于原书中错误和叙述不够准确的地方，也做了修改。另外，考虑到一些学生对较复杂的算法感觉理解困难，也顺便补充了一些算法的注释内容。

本书讨论的典型数据结构问题包括线性表、堆栈、队列、串、数组、递归、广义表、树、二叉树、图、排序、查找等。对于线性表、堆栈、队列、串、数组、广义表、树、二叉树和图等基本数据结构问题，详细讨论了各自的逻辑结构、存储结构以及各种算法的设计方法。排序和查找是两个应用广泛的算法设计问题，本书讨论了几种典型的排序算法，讨论了静态查找、动态查找和哈希查找的存储结构和查找方法。广义表、树、二叉树和图这些非线性结构的算法经常要设计成递归算法，本书专设一章讨论递归算法的设计方法等问题。

数据结构课程是一门理论和实践结合密切的课程。本书理论叙述简洁准确、实践应用举例丰富完整，理论通过丰富、完整的设计实例予以说明，设计实例从侧面解释了概念和应用方法，从而达到理论和实践密切结合的教学目的。本书采用C语言描述算法。

本书具有如下特点。

（1）内容丰富，难度适中，文字简洁准确，图文并茂。

（2）本书的所有算法都经上机调试通过，包括各章的操作实现函数、各章的程序设计实例以及习题解答中给出的算法设计。

（3）习题全面，覆盖面广，择要解答。每章最后设计了大量的习题，覆盖了各章的全部教学内容，并在附录B中给出了部分习题解答。

（4）课内上机参考资料丰富。数据结构课程是一门理论结合实践的课程，通常要求包含10课时以上的课内上机实习（或称项目设计）。本书各章的习题部分都专门设计了一定数量的上机实习题。另外，附录A还给出了上机实习报告内容规范和一个上机实习报告书写实例，可供学生参考。

根据作者的经验，使用本教材授课约需54～80课时，其中包括约10课时的课内上机实习。

使用本书的读者可登录华信教育资源网（www.hxedu.com.cn）免费下载教学课件。

作　者

目　录

第1章 绪 论

计算机是对各种各样数据进行处理的机器。要对数据进行处理，首先就要对数据进行有效的组织。因此，在计算机中如何有效地组织数据和高效地处理数据就是计算机科学的基本研究内容，也是继续深入学习后续课程的基础。本章主要对数据结构课程学习中将遇到的基本概念做概括性的叙述，这些内容将贯穿数据结构课程的整个学习过程。

本章内容主要包括：数据结构的基本概念、抽象数据类型的概念和意义、算法和算法的时间复杂度。

1.1 数据结构的基本概念

1.1.1 数据、数据元素、数据元素的数据类型

数据是人们利用文字符号、数字符号以及其他规定的符号对现实世界的事物及其活动所做的抽象描述。

例如，“今天天气情况是，最高温度为5℃，最低温度为-5℃”，就是关于今天天气情况的描述数据。又如，“班上甲同学姓名叫张三，乙同学姓名叫李四”，就是关于班上同学姓名的描述数据。

表示一个事物的一组数据称作一个**数据元素**。构成数据元素的数据称作该数据元素的**数据项**。

例如，要描述学生信息，可包括学生的学号、姓名、性别、年龄等数据。学生的学号、姓名、性别、年龄等数据构成学生情况描述的数据项，包括学号、姓名、性别、年龄等数据项的一组数据构成学生信息的一个数据元素。表1-1是一个包含3个数据元素的学生信息表。

表1-1 学生信息表

学号	姓名	性别	年龄
2000001	张三	男	20
2000002	李四	男	21
2000003	王五	女	22

在讨论数据结构时，关于数据元素、数据项的描述都需要使用某种高级程序设计语言来描述，本书采用C语言描述。学生信息的数据元素是由多个数据项构成的，其数据元素的数据类型的C语言描述方法定义为如下结构体：

```
struct Student
{
    long  number;
    char  name[10];
    char  sex[3];
    int  age;
};
```

通过上述定义，用户自定义的结构体struct Student就可像C语言中基本数据类型char、int、float一样使用。

为使用简便起见，还可以把上述定义改写为：

```
typedef struct Student
{
    long  number;
    char  name[10];
    char  sex[3];
    int  age;
} StudentType;
```

经过上述定义后，StudentType 就被看作和 struct Student 含义相同的标识符。

上述学生情况数据元素的数据类型定义为 StudentType，这与把数据元素定义为 int、 float、long、char 等数据类型一样，都是给出了具体数据元素的数据类型。

但是，像数学一样，数据结构课程讨论的大部分算法都可以适用于任何数据类型的数据元素。这时，为了考虑算法的通用性，算法要处理的数据元素也可以是不具有实际含义的数据元素。我们把没有实际含义的数据元素称作**抽象数据元素**。在本书中，抽象数据元素用 $a_0, a_1, \cdots, a_{n-1}$ 表示。

在设计算法时，任何数据元素都要指定数据类型，抽象数据元素也不例外。本书用符号 DataType 表示抽象数据元素的数据类型。当软件设计问题具体确定时，抽象数据元素的数据类型将被具体数据元素的数据类型取代。例如，若线性表的数据元素类型为 DataType，当设计的具体线性表的数据元素类型为 int 时，可通过预先定义 DataType 为 int 来完成程序的设计；当设计的具体线性表的数据元素集合为表 1-1 中的学生信息时，可通过定义 DataType 为 StudentType 来完成程序的设计。具体的 C 语言语句为：

```
    typedef  int  DataType;
或  typedef  StudentType  DataType;
```

1.1.2 数据的逻辑结构

数据元素之间的相互联系方式称为**数据的逻辑结构**。

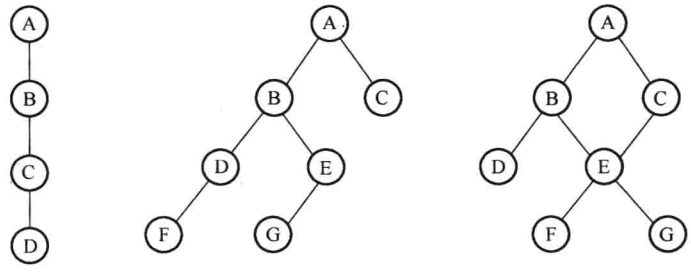
按照数据元素之间的相互联系方式，数据的逻辑结构主要可分为线性结构、树状结构和图形结构三种。

线性结构的定义是：除第一个和最后一个数据元素外，每个数据元素只有一个唯一的前驱数据元素和一个唯一的后继数据元素。线性结构可以表示为如图 1-1（a）所示的形式，图中，A、B、C、D 为数据元素，A 是第一个数据元素，D 是最后一个数据元素，A 是 B 的前驱数据元素，C 是 B 的后继数据元素；依次类推。

树状结构的定义是：除根结点外，每个数据元素只有一个唯一的前驱数据元素，可有零个或若干个后继数据元素。图 1-1（b）是一个树状结构的例子。对于数据元素 A、B、C、D、E、F、G，数据元素 A 是根结点，A 没有前驱数据元素，有两个后继数据元素 B 和 C；数据元素 B 的前驱数据元素为 A，后继数据元素为 D 和 E；数据元素 C 的前驱数据元素为 A，没有后继数据元素；如此等等。

图形结构的定义是：每个数据元素可有零个或若干个前驱数据元素和零个或若干个后继数据元素。图 1-1（c）是一个图形结构的例子。对于数据元素 A、B、C、D、E、F、G，若以 A 为起始点，则数据元素 E 有两个前驱数据元素 B 和 C，有两个后继数据元素 F 和 G。

树状结构和图形结构也可以归为非线性结构。数据元素之间不存在如图 1-1（a）所示的一对一关系的结构都称为非线性结构。



（a）线性结构　　（b）树状结构　　（c）图形结构

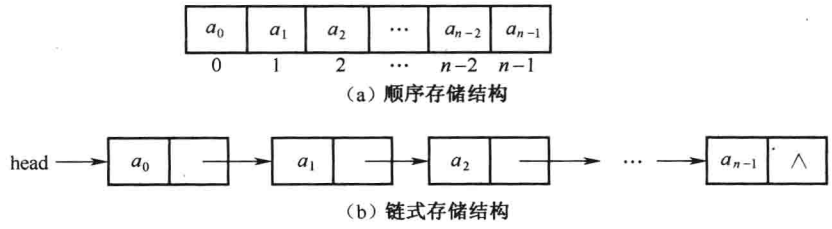
图 1-1　基本的数据逻辑结构

1.1.3　数据的存储结构

任何需要计算机进行管理和处理的数据元素都必须首先按某种方式存储在计算机中。数据元素在计算机中的存储方式称为**数据的存储结构**。数据存储结构的基本形式有两种：一种是顺序存储结构，另一种是链式存储结构。

顺序存储结构是指把数据元素存储在一块连续地址空间的内存中。其特点是，逻辑上相邻的数据元素在物理上也相邻，数据间的逻辑关系表现在数据元素的存储位置关系上。当采用高级程序设计语言表示时，实现顺序存储结构的方法是使用数组。如图 1-2（a）所示为线性结构数据元素 $a_0, a_1, \cdots, a_{n-1}$ 的顺序存储结构示意图。其中，0, 1, 2, …, n−2, n−1 既是数据元素的编号，也是存储数据元素 $a_0, a_1, \cdots, a_{n-1}$ 的数组下标。

指针是指向物理存储单元地址的变量。我们把由数据元素域和指针域组成的一个结构体称为一个结点。**链式存储结构**使用指针把相互直接关联的结点（即直接前驱结点或直接后继结点）链接起来。其特点是，逻辑上相邻的数据元素在物理上（即内存存储位置上）不一定相邻，数据间的逻辑关系表现在结点的链接关系上。如图 1-2（b）所示为线性结构数据元素 $a_0, a_1, \cdots, a_{n-1}$ 的链式存储结构示意图。其中，上一个结点到下一个结点的箭头表示上一个结点的指针域中保存的下一个结点在内存中的存储地址。head 是指向第一个结点的指针，通常称为头指针。



（a）顺序存储结构

（b）链式存储结构

图 1-2　数据存储结构的两种基本形式

顺序存储结构和链式存储结构是两种最基本、最常用的存储结构。除此之外，利用顺序存储结构和链式存储结构进行组合，还可以有一些更复杂的存储结构。

1.1.4　数据的操作

一种数据类型数据允许进行的某种操作称作**数据的操作**，一种数据类型数据所有的操作称作**数据的操作集合**。

数据结构课程在讨论数据的操作时，一般从抽象和具体两个角度进行讨论。在抽象角度下，数据的操作主要讨论数据操作所完成的逻辑功能，这部分内容一般与数据的逻辑结构一起讨论；在具体角度下，数据的操作主要讨论数据操作的具体实现算法。例如，若某软件要对表 1-1 中的

学生信息进行处理，对学生信息可能进行的操作有：插入一条数据元素，删除一条数据元素，列出所有数据元素的值等。所以，该问题数据的操作有：插入一条数据元素，删除一条数据元素，列出所有数据元素的值等。在其抽象角度下，这些操作的逻辑功能如其字面含义所述；在具体角度下，表 1-1 的学生信息既可采用图 1-2（a）的顺序存储结构存储数据元素，也可采用图 1-2（b）的链式存储结构存储数据元素，不同的存储结构操作实现的具体算法将不同。

1.1.5 “数据结构”课程讨论的主要内容

“数据结构”课程主要讨论表、堆栈、队列、串、数组、树、二叉树、图等典型的常用数据结构，在讨论这些典型的常用数据结构时，主要从它们的逻辑结构、存储结构和数据操作 3 个方面进行分析讨论。例如，我们在第 2 章中讨论线性表时，2.1 节将讨论线性表的抽象数据类型（即线性表的逻辑结构和逻辑结构意义下的操作功能），2.2 节将讨论线性表的顺序存储结构和顺序存储结构下各基本操作的具体实现算法，2.3 节将讨论线性表的链式存储结构和链式存储结构下各基本操作的具体实现算法。其他各章中对堆栈、队列、串、数组、树、二叉树、图等进行讨论的各节安排次序与第 2 章的类同。

1.2 抽象数据类型

类型是一组值的集合。

例如，int 类型就是具体计算机所能表示的 int 类型数值的集合。通常，int 类型的数值范围是−32768～32767。又如，float 类型就是具体计算机所能表示的 float 类型数值的集合。

数据类型是指一个类型和定义在这个类型上的操作集合。

例如，当我们说计算机中的 int 数据类型时，我们不仅指 int 类型所能表示的−32768～32767 的数值范围，还指 int 类型的数据允许进行的加（+）、减（−）、乘（*）、除（/）和求模（%）操作。

在“数据结构”课程中，通常把在已有的数据类型基础上设计新的数据类型的过程称作数据结构设计。

抽象数据类型（Abstract Data Type，ADT）是指一个逻辑概念上的类型和这个类型上的操作集合。

从定义看，数据类型和抽象数据类型的定义基本相同。数据类型和抽象数据类型的不同之处仅仅在于：数据类型通常指的是高级程序设计语言支持的基本数据类型，而抽象数据类型指的是在基本数据类型支持下用户新设计的数据类型。“数据结构”课程主要讨论表、堆栈、队列、串、数组、树、二叉树、图等典型的常用数据结构，这些典型的常用数据结构就是一个个不同的抽象数据类型。

盖楼时，如果用砖、水泥、沙子来盖，则不仅建造周期长，而且楼不可能盖得很高（否则将不安全）；如果用水泥预制板（即更大的模块）来盖，则不仅建造周期短（水泥预制板由专门的公司按规范的规格提供），楼能盖很高，而且所建造的高楼能保证安全。从数学的观点看，水泥预制板使高楼建造过程的接缝数量大大减少，从而大大降低了高楼建造的复杂度。

抽象数据类型使软件设计成为工业化流水线生产的一个中间环节。一方面，根据给出的抽象数据类型定义，负责设计这些抽象数据类型的专门公司（或专门设计人员）设计该抽象数据类型的具体存储结构，以及在具体存储结构下各操作的具体实现算法；另一方面，利用已设计实现的抽象数据类型模块，负责设计应用软件的专门公司（或专门设计人员）可以安全、快速、方便地完成该应用软件系统的设计。这样的方法与使用水泥预制板建造高楼的方法类同。水泥预制板规格的设计是高楼建造的一个中间环节。一方面，根据给出的水泥预制板规格，负责提

供水泥预制板的专门公司建造水泥预制板；另一方面，根据给出的水泥预制板规格，高楼的设计人员和建造人员可以安全、快速、方便地完成高楼的设计和建造。

软件的设计采用模块化方法，抽象数据类型（如线性表、堆栈、队列、串、数组、广义表、树、二叉树、图等）就是构造大型软件的最基本模块。用这些已有专门公司设计好的抽象数据类型，就可以安全、快速、方便地设计功能复杂的大型软件。

数据结构课程讨论线性表、堆栈、队列、串、数组、广义表、树、二叉树、图等基本数据结构的功能和设计方法。在大部分高级程序设计语言的类库包中，这些常用的数据结构都已设计完成。设计人员通常不需要重新设计和实现这些数据结构，只需要根据问题的要求，首先把相应的数据结构头文件包含进来，然后定义具体的变量，调用相应的函数，即可实现所要完成的功能。从这个角度看，有些人可能会觉得“数据结构”课程的学习没有必要。对于计算机学科的学生，以及那些希望深入学习掌握计算机软件设计方法的学生来说，本课程对典型数据结构的分析以及对典型算法设计方法的讨论和训练，既能帮助学生打好软件设计的扎实基础，也能帮助学生掌握软件模块化设计的基本方法。

1.3 算法和算法的时间复杂度

1.3.1 算法

算法是描述求解问题方法的操作步骤集合。

算法要用某种语言来描述。描述算法的语言主要有三种形式：文字形式、伪码形式和程序设计语言形式。文字形式是指用中文或英文这样的文字来描述算法。伪码形式是指用一种仿程序设计语言的语言（因为这样的描述语言不是真正的程序设计语言，所以称作伪码）来描述算法。程序设计语言形式是指用某种高级程序设计语言来描述算法。用高级程序设计语言描述算法的优点是，算法描述既简洁易读，又可以直接输入计算机调用或运行。本书采用 C 语言这种高级程序设计语言描述算法。

下面给出算法设计的两个例子。从这两个例子，读者可体会用高级程序设计语言描述算法的基本方法以及这种描述方法的优点。

【例 1-1】 设计一个把存储在数组中的有 n 个抽象数据元素 $a_0, a_1, \cdots, a_{n-1}$ 逆置的算法。逆置就是把数据元素序列 $a_0, a_1, \cdots, a_{n-1}$ 变换为数据元素序列 $a_{n-1}, \cdots, a_1, a_0$，并要求原数组中的数据元素值不被改变。

【算法参数设计】 这个算法的参数应该包括三个：表示原数组的输入参数 a，表示数据元素个数的输入参数 n，表示逆置后数组的输出参数 b。

算法设计如下：

```
void Reverse(DataType a[], int n, DataType b[] )
{
    int i;
    for(i = 0; i < n; i++)
        b[i] = a[n - 1 - i];
}
```

该算法共进行 n 次赋值，算法的实现过程如图 1-3 所示。

【例 1-2】 设计一个把存储在数组中的有 n 个抽象数据元素 $a_0, a_1, \cdots, a_{n-1}$ 就地逆置的算法。就地逆置就是把数据元素序列 $a_0, a_1, \cdots, a_{n-1}$ 变换为数据元素序列 $a_{n-1}, \cdots, a_1, a_0$，并要求这种变换在原数组中进行。

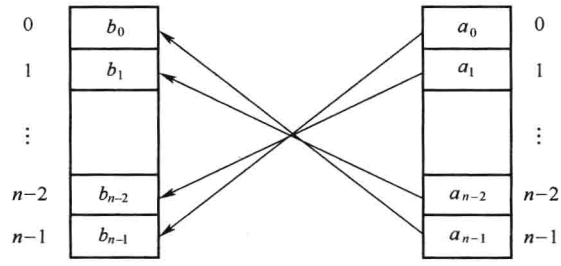


图 1-3　逆置算法实现方法

【算法参数设计】　这个算法的参数应该包括两个：表示原数组和就地逆置后数组的参数 a，表示数据元素个数的参数 n。

算法设计如下：

```
void Reverse(DataType a[], int n)
{
    int i, m = n/2;
    DataType temp;

    for(i = 0; i < m; i++)                    //进行 m 次调换
    {
        temp = a[i];
        a[i] = a[n - 1 - i];
        a[n - 1 - i] = temp;
    }
}
```

注意：在 C 语言中，当除数和被除数都是整数时，运算符“/”表示整数相除，即商只取整数部分。例如，当 n=10 时，n/2 的运算结果为 5，即变量 m 得到赋值 5；当 n=11 时，n/2 的运算结果仍为 5，即变量 m 仍得到赋值 5。

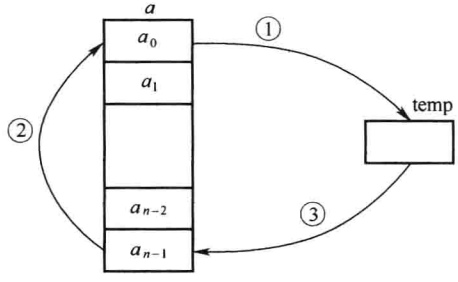


图 1-4　就地逆置算法实现方法

该算法共进行 $n/2$ 次调换，第一次调换的过程如图 1-4 所示。

1.3.2　算法的性质和设计目标

任何算法设计都应满足以下性质。

① 输入性：具有零个或若干个输入量。

② 输出性：至少产生一个输出量或执行一个有意义操作。

③ 有限性：执行语句的序列是有限的。

④ 确定性：每条语句的含义明确，无二义性。

⑤ 可执行性：每条语句都应在有限的时间内完成。

用高级程序设计语言描述算法时，一个算法就是一个函数。算法的输入量就是函数的输入参数，算法的输出量就是函数的输出参数。由于高级程序设计语言规范了语句格式，不允许二义性语句，因此在用高级程序设计语言语句组合出的算法中，只要没有无限循环，则必然满足算法的有限性、确定性和可执行性的性质。

应用程序通常是通过调用函数（即算法）来完成的。程序和算法的唯一区别是，程序允许

无限循环，而算法不允许无限循环。构成无限循环的一组语句如下：

```
while(1)                                    //循环条件永真
{
    …                                       //任意语句序列
}
```

算法设计应满足以下 5 个目标。

① 正确性。算法应确切地满足具体问题的需求。这是算法设计的基本目标。

② 可读性。算法的可读性有利于人们对算法的理解，这既有利于程序的调试和维护，也有利于算法的交流和移植。算法的可读性主要体现在两个方面：一是变量名、常量名、函数名等的命名要见名知意；二是要有足够多的注释。

③ 健壮性。当输入非法数据时，算法要能做出适当的处理，而不应产生不可预料的结果。

④ 高时间效率。算法的时间效率是指运行算法需要花费时间的多少。对于同一个问题，如果有多个算法可供选择，则应尽可能选择运行时间短的算法。运行时间短的算法称作高时间效率的算法。

⑤ 高空间效率。算法的空间效率是指运行算法需要占用的额外内存空间的多少。对于同一个问题，如果有多个算法可供选择，则应尽可能选择占用内存空间少的算法。占用内存空间少的算法称作高空间效率的算法。

算法的高时间效率和高空间效率通常是矛盾的。例如有些问题，若算法占用了较多的内存空间，则算法只需要进行较少次循环就能实现，因而时间效率会提高；若算法占用了较少的内存空间，则算法需要进行较多次循环才能实现，因而时间效率会降低。在目前计算机内存价格快速下降的趋势下，当算法设计的时间效率目标和空间效率目标发生矛盾时，对于大多数情况来说，算法的时间效率目标应首先被考虑。

1.3.3 算法的时间效率分析

算法的执行时间需通过根据该算法编制的程序在计算机中运行所消耗的时间来度量。度量一个程序在计算机中的执行时间通常采用如下两种方法。

（1）事后统计方法。方法是，设计一组或若干组测试数据，然后分别运行根据不同的算法编制的程序，并比较这些程序的实际运行时间，从而确定算法时间效率的优劣。这种方法有两个缺陷：一是必须实际运行依据算法编制的程序，而这通常是比较麻烦和费时的；二是有些算法的测试数据设计困难，因为不同的算法对不同的测试数据，测试结果可能不同，要设计出能客观、全面反映算法时间效率的测试数据有时很困难。

（2）事前分析方法。方法是，用数学方法直接对算法的时间效率进行分析，不需要实际运行算法。这种方法在实际中比较常用。

根据算法编制的程序在计算机中运行所消耗的时间与下列因素有关：

- 书写算法的程序设计语言；
- 编译产生的机器语言代码的质量；
- 机器执行指令的速度；
- 问题的规模，即算法的耗时与算法所处理数据个数 n 的函数关系。

在这 4 个因素中，前 3 个都与具体的计算机有关。分析算法的时间效率应抛开具体的计算机，仅考虑算法本身的因素。因此，事前分析方法主要通过分析算法的耗时与算法所处理数据个数 n 的函数关系，来评估一个算法的优劣。

算法的耗时与算法所处理数据个数 n 的函数关系的分析称作**算法的时间效率分析**。算法的

时间效率分析主要是分析算法的耗时与算法所处理数据个数 n 的**数量级意义上的函数关系**。因此，算法的时间效率分析也称作算法的时间复杂度分析。

算法的时间复杂度分析通常采用 $O(f(n))$表示法（$O(f(n))$读作大 O 的 $f(n)$）。

【定义】 $T(n) = O(f(n))$当且仅当存在正常数 c 和 n_0，对所有的 $n(n \geqslant n_0)$满足 $T(n) \leqslant cf(n)$。

由于上述定义中对所有的 $n(n \geqslant n_0)$条件，只要 n 比较大时一般均成立，而我们考虑的算法的时间复杂度也主要是数据个数 n 相当大时的情况，因此具体分析一个算法的时间复杂度 $T(n)$时，一般不考虑 n 为一个较小的数时 $T(n) \leqslant cf(n)$不成立的情况。

通俗地说，$O(f(n))$给出了函数 $f(n)$的上界。

令函数 $T(n)$为算法的时间复杂度，其中 n 为算法处理的数据个数，则 $T(n) = O(f(n))$表示，算法的时间复杂度随数据个数 n 的增长率与函数 $f(n)$的增长率相同，或者说两者具有相同的数量级。

当算法的时间复杂度 $T(n)$和数据个数 n 无关系时，$T(n) \leqslant c \times 1$，所以此时算法的时间复杂度 $T(n) = O(1)$；当算法的时间复杂度 $T(n)$和数据个数 n 为线性关系时，$T(n) \leqslant cn$，所以此时算法的时间复杂度 $T(n) = O(n)$；当算法的时间复杂度 $T(n)$和数据个数 n 为平方关系时，$T(n) \leqslant cn^2$，所以此时算法的时间复杂度 $T(n) = O(n^2)$；其余类推，还有 $O(n^3)$、$O(\text{lb}n)$[①]、$O(\lg n)$[②]、$O(2^n)$等。

显然，对于处理同样问题的算法来说，时间复杂度为 $O(1)$的算法优于时间复杂度为 $O(n)$的算法，时间复杂度为 $O(n)$的算法优于时间复杂度为 $O(n^2)$的算法，时间复杂度为 $O(\text{lb}n)$的算法优于时间复杂度为 $O(n)$的算法。

可见，分析一个算法中基本语句执行次数和数据个数 n 在数量级意义上的函数关系，就可以分析出该算法的时间复杂度，从而评估该算法的优劣。

下面的 4 个例题是算法时间复杂度分析的 4 种典型情况。

【例 1-3】 设表示 n 阶矩阵的数组 a 和 b 在前边部分已赋值，求两个 n 阶矩阵相乘运算算法的时间复杂度。

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
    {
        c[i][j] = 0;                                    //基本语句 1
        for(k = 0; k < n; k++)
            c[i][j] = c[i][j] + a[i][k] * b[k][j];      //基本语句 2
    }
```

【解】 设基本语句的执行次数为 $f(n)$，有

$$f(n) = c_1 n^2 + c_2 n^3$$

因为 $T(n) = f(n) = c_1 n^2 + c_2 n^3 \leqslant cn^3$，其中 c_1, c_2, c 均为常数，所以该算法的时间复杂度为 $T(n) = O(n^3)$。

【例 1-4】 求如下算法片段的时间复杂度。

```
for(i = 1; i <= n; i = 2 * i)
    printf("i = %d\n", i);                              //基本语句
```

【解】 设基本语句的执行次数为 $f(n)$，有 $2^{f(n)} \leqslant n$，即有 $f(n) \leqslant \text{lb}n$。

因为 $T(n) = f(n) \leqslant \text{lb}n \leqslant c\text{lb}n$，其中 $c = 1$，所以该算法的时间复杂度为 $T(n) = O(\text{lb}n)$。

在很多情况下，若算法中数据元素的取值情况不同，则算法的时间复杂度也会不同。此时，算法的时间复杂度应是数据元素最坏情况下取值的时间复杂度（简称为最坏时间复杂度），或数

① lb n 为求 n 的以 2 为底的对数，即 $\text{Log}_2 n$。

② lg n 为求 n 的以 10 为底的对数，即 $\text{Log}_{10} n$。

据元素等概率取值情况下的平均时间复杂度（简称为平均时间复杂度）。

【例 1-5】 下边的算法采用冒泡排序法对数组 a 中的 n 个整数类型的数据元素（a[0]～a[n-1]）从小到大进行排序，求该算法的时间复杂度。

```
void BubbleSort(int a[], int n)
{
    int i, j, flag=1;
    int temp;

    for(i = 1; i < n && flag == 1; i++)
    {
        flag = 0;
        for(j = 0; j < n-i; j++)
        {
            if(a[j] > a[j+1])
            {
                flag = 1;
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
```

【解】 这个算法的时间复杂度随待排序数据的不同而不同。当某次排序过程中没有任何两个数组元素交换位置时，表明数组元素已排序完毕，此时算法将因标记 flag=0 不满足循环条件而结束。但是，在最坏情况下，每次排序过程中都至少有一对数组元素交换位置，因此，最坏情况下该算法的时间复杂度分析如下。

设基本语句的执行次数为 $f(n)$，则在最坏情况下有

$$f(n)\approx n+4n^2/2$$

因为 $T(n)=f(n)\approx n+2n^2\leqslant cn^2$，其中 c 为常数，所以该算法的最坏时间复杂度为 $T(n)=O(n^2)$。

【例 1-6】 下边算法在一个有 n 个数据元素的数组 a 中删除第 i 个位置的数组元素，要求删除成功时数组元素个数减 1，求该算法的时间复杂度。其中，数组下标为 0～n-1。

```
int Delete(int a[], int *n, int i)
{
    int j;
    if(i < 0 || i >= *n) return 0;              //删除位置错误返回
    for(j = i + 1; j < *n; j++) a[j-1] = a[j];  //顺次移位填补
    (*n)--;                                     //数组元素个数减 1
    return 1;                                   //删除成功返回
}
```

【解】 这个算法的时间复杂度随删除数据的位置不同而不同。当删除最后一个位置的数组元素时，有 $i=n-1$，$j=i+1=n$，此时因为不需要移位填补而循环次数为 0；当删除倒数最后一个位置的数组元素时，有 $i=n-2$，$j=i+1=n-1$，此时因为只需要移位填补一次而循环次数为 1；依次类推，当删除第一个位置的数组元素时，有 $i=0$，$j=i+1=1$，此时因为需移位填补 $n-1$ 次而循环次数为 $n-1$。此时，算法的时间复杂度应是删除数据位置等概率取值情况下的平均时间复杂度。

假设删除任何位置上的数据元素都是等概率的（在一般情况下，均可做等概率假设），设 P_i 为删除第 i 个位置上数据元素的概率，则有 $P_i=1/n$。设 E 为删除数组元素的平均次数，则有

$$E=\frac{1}{n}\sum_{i=0}^{n-1}(n-1-i)=\frac{1}{n}[(n-1)+(n-2)+\cdots+2+1+0]=\frac{1}{n}\cdot\frac{n(n-1)}{2}=\frac{n-1}{2}$$

因为 $T(n)=E\leqslant(n+1)/2\leqslant cn$，其中 c 为常数，所以该算法的平均时间复杂度为 $T(n)=O(n)$。

上面的 4 个例题是算法时间复杂度分析的 4 种典型情况。很多实际问题的算法时间复杂度分析是上述几种典型情况的组合或某种典型情况的变种。

1.3.4 算法耗时的实际测试

前面比较详细地讨论了时间效率分析的事前分析方法，本节将通过实际例子，再讨论一下时间效率分析的事后统计方法，即算法耗时的实际测试方法，并结合时间效率分析的事前分析方法，具体说明两种分析方法的结合使用。

【例 1-7】 在数据元素个数为 30000 时，对比冒泡排序算法和快速排序算法的实际耗时。

根据问题的要求，设计如下测试程序，并在计算机中实际运行。

```
#include <stdio.h>
#include<stdlib.h>
#include<time.h>

typedef int KeyType;

typedef struct
{
    KeyType key;
} DataType;

void BubbleSort(DataType a[], int n)
//对数据元素 a[0]～a[n-1]进行冒泡排序
{
    int i, j, flag = 1;
    DataType temp;
    for(i = 1; i < n && flag == 1; i++)
    {
        flag = 0;
        for(j = 0; j < n-i; j++)
        {
            if(a[j].key > a[j+1].key)
            {
                flag = 1;
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
}
```