高职高专计算机实用规划教材

—— 案例驱动与项目实践

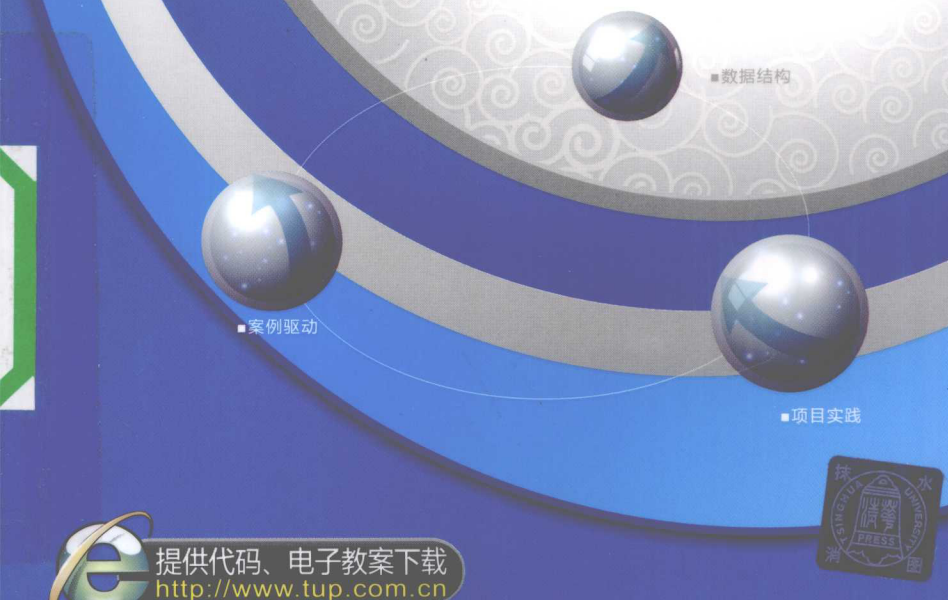
本书特色：

- 通过C/C#对数据结构与算法的描述来揭示面向过程和面向对象的思想
- 理解数据结构特性，培养计算机数据抽象能力和计算机思维能力

数据结构

(C/C#语言版)

段恩泽　肖守柏　主　编



提供代码、电子教案下载
http://www.tup.com.cn

清华大学出版社

高职高专计算机实用规划教材——案例驱动与项目实践

数据结构
(C/C#语言版)

段恩泽　肖守柏　主　编

清华大学出版社
北　京

内 容 简 介

“数据结构”是计算机及相关专业必修的核心基础课程。本书采用 C 和 C#两种语言作为算法描述的语言，对常用的数据结构与算法作了系统的介绍，力求概念清晰简单，注重实际应用。本书通过两种语言对数据结构与算法的不同描述来揭示面向过程和面向对象两种不同的思想。全书共分为 8 章，依次介绍了数据结构与算法及本书用到的数学、C 和 C#知识、线性表、栈和队列、串和数组、树型结构和图结构，以及排序和查找等基本运算。

本书主要面向高职高专院校计算机专业的学生，也可作为非计算机专业学生的选修教材及计算机应用技术人员的自学参考书。

图书在版编目(CIP)数据

数据结构(C/C#语言版)/段恩泽，肖守柏主编. —北京：清华大学出版社，2010.6
(高职高专计算机实用规划教材——案例驱动与项目实践)
ISBN 978-7-302-22506-5
Ⅰ. 数… Ⅱ. ①段…②肖… Ⅲ. ①数据结构—高等学校：技术学校—教材②C 语言—程序设计—高等学校：技术学校—教材 Ⅳ. ①TP311.12②TP312
中国版本图书馆 CIP 数据核字(2010)第 068303 号

责任编辑：黄　飞　桑任松
装帧设计：杨玉兰
责任印制：何　芊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京富博印刷有限公司
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185×260　**印　张**：18.75　**字　数**：453 千字
版　　次：2010 年 6 月第 1 版　　**印　　次**：2010 年 6 月第1 次印刷
印　　数：1～4000
定　　价：28.00 元

产品编号：035232-01

前　言

自从美国唐·欧·克努特教授用汇编语言编写的《计算机程序设计技巧》第一卷《基本算法》问世以来，已经出现了用 Pascal、Java、C、C++、C#等语言编写的数据结构方面的书。总体说来，这些语言基本上分为面向过程的语言和面向对象的语言两大类，也出现过采用两种语言描述数据结构的书籍，如 C 和 C++语言描述，但作者实际上是按照 C++语言描述。同时采用面向过程和面向对象语言描述数据结构，目前国内基本上是空白。对于同一种数据结构与算法，同时采用面向过程和面向对象语言进行描述，可以从中更深刻理解这两种思想的不同，这对于深刻理解计算机语言和思想有着重要的作用。C 语言是现在最流行的面向过程的语言，在业界使用非常广泛。而 C#语言作为微软在新一代开发平台(.NET)上推出的、完全面向对象的语言，凭着其简洁、高效、模板、标准化的特性，使得 C#语言像程序设计语言中的一件艺术品，也吸引着越来越多的开发人员。当然，C#语言也吸收了 C 语言的一些东西，如语法等，所以，在有些方面，C#与 C 是相似的。鉴于此，编者决定编写本书，使用 C 和 C#语言来描述数据结构与算法。

本书共分为 8 章，第 1 章介绍了数据结构和算法的基本概念及本书用到的数学知识、C 语言知识和 C#语言的知识；第 2～6 章分别讨论了线性表、栈、队列、串、数组、树、二叉树及图等常用的数据结构及其应用；第 7、8 两章分别讨论了排序和查找常用的各种方法及其应用。

由于本书采用 C 和 C#两种语言对数据结构进行描述，为节省篇幅，本书对内容的处理如下：先对数据结构进行分析，如数据结构的概念、逻辑结构、物理结构等，然后给出每种数据结构的两种语言描述。这样，把重点放在了数据结构本身上，而不只是考虑其语言实现。这体现了“数据结构”课程的目的，即理解数据结构的特性，培养计算机的数据抽象能力和计算机思维能力。

本书由成都东软信息技术职业学院段恩泽、江西蓝天学院肖守柏两位老师主编，江西蓝天学院蔡爱平、江西吉安市信息化工作办公室习爱民两位老师共同完成。其中，C 语言部分，第 1、2 章和第 3～5 章分别由肖守柏、蔡爱平两位老师编写；第 6～8 章由习爱民老师编写。C#语言部分由段恩泽老师编写，全书由段恩泽老师统稿、整理。

本书主要面向高职高专院校计算机专业的学生，也可作为非计算机专业学生的选修教材及计算机应用技术人员的自学参考书。

本书在编写过程中得到了清华大学出版社黄飞等老师的大力支持，他们为本书的修订和出版做了大量的工作，在此表示感谢。

尽管编者在写作过程中非常认真和努力，但由于编者水平有限，特别是分别采用面向过程语言和面向对象语言来描述数据结构，是一种新的尝试，书中难免存在错误和不足之处，恳请广大读者和专家批评指正。如果您对本书有什么意见、问题或想法，欢迎您通过下面的邮箱告知编者，编者将不胜感激：

Email：duanez@neusoft.com，Xiaoshoubai1015@163.com

其中，与C#语言相关的问题发到前一个邮箱，与C语言相关的问题发到后一个邮箱，并请在邮箱的主题栏中注明：数据结构(C/C#)。

编　者

目　录

第1章 绪 论

数据是外部世界信息表示的计算机化，是计算机加工处理的对象。运用计算机处理数据时，必须解决 4 个方面的问题：一是如何在计算机中方便、高效地表示和组织数据；二是如何在计算机存储器(内存和外存)中存储数据；三是如何对存储在计算机中的数据进行操作，可以有哪些操作，如何实现这些操作，如何对同一问题的不同操作方法进行评价；四是必须理解每种数据结构的性能特征，以便选择一种适合于一个特定问题的数据结构。这些问题就是数据结构这门课程所要研究的主要问题。本章首先说明学习数据结构的必要性和本书的目的，然后解释数据结构及其有关的概念，接着讨论算法的相关知识，最后简单介绍本书所要用到的相关数学知识、C 语言知识和 C#知识。

1.1 数 据 结 构

1.1.1 学习数据结构的必要性

众所周知，虽然每个人都懂得英语的语法与基本类型，但是对于同样的作文题目，每个人写出的作文，水平却高低不一。程序设计也和写英语作文一样，虽然程序员都懂得程序的语法与语意，但是对于同样的问题，程序员写出来的程序却不一样。有的人写出来的程序效率很高，有的人却用最复杂的方法来解决一个简单的问题。

当然，程序设计能力的提高仅仅靠看几本程序设计书是不行的。“师父领进门，修行看个人”，只有多思索、多练习，才能提高自己的程序设计水平；否则，书看得再多，提高也不大。记得刚学程序设计时，常听人说要想提高程序设计水平，最重要的还是多看别人写的程序，多去思考问题。从看别人写的程序中，可以发现出效率更高的解决方法；从思考问题的过程中，可以了解解决问题的方法常常不只一个。如何运用先前解决问题的经验，来解决更复杂、更深入的问题，是提高程序设计能力的最有效途径。

数据结构正是前人在思索问题的过程中所想出的解决方法。一般而言，在学习程序设计一段时间后，学习“数据结构”便是为了让你的程序设计的能力上一个台阶。如果只学会了程序设计的语法和语意，那么只能解决程序设计 1/3 的问题，而且运用的方法并不是最有效的。但如果学会了数据结构的概念，你就能在程序设计上，运用最有效的方法来解决八成以上的问题。

我们知道，数据结构不仅仅是计算机专业的核心课程之一，而且是其他非计算机专业的主要选修课程之一。数据结构的研究不仅涉及计算机硬件的研究范围，而且和计算机软件的研究也有着密切的联系，无论是编程还是操作系统，都涉及如何在存储器中为数据元素分配存储空间、如何组织数据，以便更方便地处理数据元素。《数据结构》这门课程的目的有 3 个。第一个是讲授常用的数据结构，这些数据结构形成了程序员基本数据结构工

具箱(Toolkit)。对于许多问题，工具箱里的数据结构是理想的选择。就像 Microsoft .NET Framework 的 Windows 应用程序开发中的工具箱，程序员可以直接拿来或经过少许的修改就可以使用，非常方便。第二个是讲授常用的算法，这和数据结构一样，程序员可以直接拿来或经过少许的修改就可以使用，并且通过算法训练来提高编程能力。第三个目的是通过程序设计的技能训练促进学生综合能力的提高。

1.1.2 基本概念和术语

在本小节中，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

数据(Data)是外部世界信息的载体，是客观事物的符号表示，是所有能输入到计算机中并被计算机程序处理的符号的总称。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。计算机程序可以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. 数据元素和数据项

数据元素(Data Elememt)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(Data Item)组成。数据项是不可分割的、含有独立意义的最小数据单位，数据项有时也称为字段(Field)或域。例如，在数据库管理系统中，数据表中的一条记录就是一个数据元素。这条记录中的学生学号、姓名、性别、籍贯、出生年月、成绩等字段就是数据项。数据项分为两种，一种叫做初等项，如学生的性别、籍贯等，在处理时不能再进行分割；另一种叫做组合项，如学生的成绩，它可以再分为数学、物理、化学等更小的项。

3. 数据对象

数据对象(Data Object)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{0,±1,±2,±3,…}，字符数据对象是{a,b,c,…}等。

在程序中，数据对象在程序运行时存在。有些数据对象是由系统定义的，如运行期堆栈、子程序活动记录、文件缓冲区及空闲区等。这些数据对象是在程序运行过程中按照需要自动创建的。有些数据对象是由程序员定义的，如程序员通过程序中的声明和语句显示创建的变量、常量、数组和文件等。一个数据对象代表了能存放和检索数据值的容器，它有许多属性，其中最重要的属性是类型、位置、值、名等。数据对象分简单数据对象和结构化数据对象，简单数据对象的操作可以由计算机硬件实现，如各种整型变量、字符变量。结构化数据对象的操作由软件模拟实现，如数组对象、用户自定义的数据类型变量。

4. 数据类型

数据类型(Data Type)是高级程序设计语言中的概念，是数据的取值范围和对数据进行操作的总和。对于数据的取值范围而言，不论是 C 语言还是 C#语言，都与计算机硬件有关。

比如，对于 32 位的机器而言，一个整型都是 4 个字节，表示范围为-2^{31}～$2^{31}-1$，即-2147483648～2147483647。而整数类型的数据允许施加的操作通常有：单目取正或取负运算，双目加、减、乘、除、取模等运算。但是 C#语言是面向对象的语言，对数据类型的处理与 C 语言不同。C#语言把整数类型表示成一个类，对数据进行了封装，把对整数类型的数据施加的操作封装在该类中，由类名或对象名调用操作。

在通用的计算机高级语言中，一般都有整型、实型、字符型、数组、枚举、结构体等数据类型。数据类型可分为两类：一类是非结构的原子类型，如整型、实型、字符型等基本类型；另一类是结构类型，它的成分可以由多个结构类型组成，并可以分解。结构类型的成分可以是非结构的，也可以是结构的。例如，数组的成分可以是整型等基本类型，也可以是数组等结构类型。

C#语言的数据类型比 C 多。C#不仅有字符串这种基本的数据类型，还有类、委托、接口、事件等数据类型。其中原因在于两者出现的背景不同。C 语言是随着 UNIX 操作系统的出现而流行起来的，主要用来书写系统代码，应用在底层上，速度快、效率高。然而，C 语言存在一些缺陷，如类型检查机制相对较弱、缺少支持代码重用的语言结构等，造成用 C 语言开发大程序比较困难。而 C#语言是 Microsoft .NET Framework 的首选语言，在 Internet 的大环境下出现的，为了编写面向网络的应用。所以，C#语言的抽象程度更高，应用范围更广，能够开发大程序，因此，C#语言的数据类型的种类更多。

5. 抽象数据类型

抽象数据类型(Abstract Data Type，ADT)是指一些数据以及对这些数据所进行的操作的集合。数据之间有一定的关系，不同的抽象数据类型的数据之间的关系不同。对数据进行的操作由数据间的关系决定。数据间的关系不同，对数据的操作也就不同。这些操作既向程序的其余部分描述了这些数据是怎么样的，也允许程序的其余部分改变这些数据。一个 ADT 可能是一个图形窗体以及所有能影响到该窗体的操作，也可以是一个文件以及对这个文件进行的操作，或者是一张保险费率表及相关操作等。

传统的教科书在讲到抽象数据类型时，总会用一些数学中的概念进行描述。典型的描述是："你可以把抽象数据类型想成一个定义有一组操作的数学模型。"这种描述给人的感觉好像你从不会真正用到抽象数据类型似的——除非拿它来催眠。

把抽象数据类型解释得这么空洞是完全丢了重点。抽象数据类型可以让你像在现实世界中一样操作实体，而不必在底层的实现上摆弄实体，如不用再向链表中插入一个结点了，而是可以在电子表格中添加一个数据单元，或向一组窗体类型中添加一个新类型，或给火车模型加挂一节车厢。也就是说，抽象数据类型使你可以在问题域中思考问题，而不管底层是如何实现的。这就是"抽象数据类型"概念中"抽象"的含义。也就是说，它把底层的信息隐藏起来，修改底层信息不会影响该抽象数据类型的使用，因为它提供的公有操作没有改变。并且，如果数据类型发生改变，也只需在一处修改而不会影响到整个程序。

对于抽象数据类型，有一点需要特别注意，抽象数据类型的使用者只能通过定义在抽象数据类型中的操作来使用和操作抽象数据类型中的数据，而不能直接操作这些数据，也不能通过没有定义的操作来使用和操作这些数据。从这一点可以看出，抽象数据类型实际上是把信息隐藏和封装的概念扩展到程序员自定义的数据。C#语言由于是面向对象的语言，

所以可以提供对抽象数据类型很好的支持。在 C 语言提供的类型定义机制中，因为在定义新变量时，仅需要提供类型名字，所以使定义变量变得简单。但是，该类型对象的内部结构没有被封装，任何定义了该类型的变量的子程序，都能够存取和操作该数据对象的各个成分。因此，C 语言对抽象数据类型的支持没有 C#语言好。

面向对象语言自动支持对同一 ADT 的多份实例的处理。C#的类等类型可以实现 ADT，通过构造器创建多个实例。但如果在 C 语言这样的非面向对象的环境下汇总工作，就必须自己手工实现支持处理多个实例的技术。一般来说，这意味着要为 ADT 添加一些用来创建和删除实例的服务操作，同时需要重新设计 ADT 的其他服务操作，使其能够支持多个实例。

这里需要说明的是，对于使用面向对象语言的程序员而言，首先应考虑 ADT，而后才考虑类，这是一个“深入一种语言编程”而不是“在一种语言上编程”。“在一种语言上编程”的程序员将他们的思想限制于“语言直接支持的那些构件”。如果语言工具是初级的，那么程序员的思想也是初级的。“深入一种语言编程”的程序员首先决定他要表达的思想是什么，然后决定如何使用特定语言提供的工具来表达这些思想。

一个抽象数据类型由两部分组成：数据元素集合、数据操作集合，其语法格式为：

```
ADT 抽象数据类型名字{
    数据元素：
        <数据描述>
    数据操作：
        <数据操作描述>
}
```

其中，数据描述中包含数据之间关系的描述。数据元素集合可以用文字或数学知识进行描述，这与计算机语言无关。但是，C 语言和 C#语言描述数据操作集合的形式不同。C 语言使用函数表示数据操作，C#语言中用方法表示数据操作(C#语言中的方法相当于 C 语言中的函数)。C 语言的函数声明由函数名、参数和返回值类型 3 个部分组成，C#语言的方法声明由方法名、参数、返回值类型和访问属性 4 个部分组成，其声明中多了访问属性，这是由于 C#语言是面向对象语言的缘故。C 语言的函数的参数和返回值类型都可以是指针类型，但 C#中没有指针这种概念。C 语言和 C#语言函数的参数和返回值类型都可以是引用类型，但这两种语言对引用的处理不同，C 语言中使用“&”符号表示引用，C#语言中没有这样表示。最大的不同是，如果 C 语言要对某个对象进行操作，必须把该对象作为一个参数声明在函数的参数列表中。而 C#语言是由对象直接调用该方法，所以在方法声明的参数列表中没有对象这个参数。因此，这两种语言描述数据操作集合的形式不同。在本书中规定，C 语言使用函数声明的集合描述数据操作集合，C#语言使用接口描述数据操作集合。

6. 数据结构

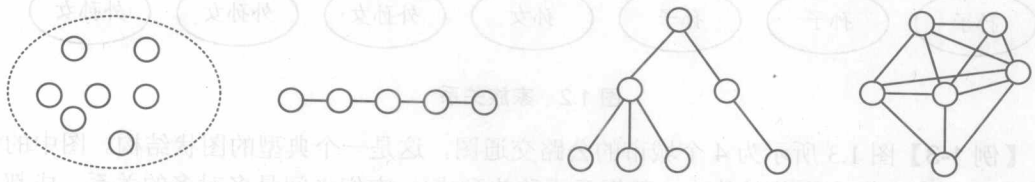
数据结构(Data Structure)是相互之间存在一种或多种特定关系的数据元素的集合。在任何问题中，数据元素之间都不是孤立的，而是存在着一定的关系，这种关系称为结构(Structure)。根据数据元素之间关系的不同特性，通常有以下 4 类基本数据结构。

(1) 集合(Set)：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，如图 1.1(a)所示。

(2) 线性结构(Linear Structure)：该结构中的数据元素存在着一对一的关系，如图 1.1(b)所示。

(3) 树形结构(Tree Structure)：该结构中的数据元素存在着一对多的关系，如图 1.1(c)所示。

(4) 图状结构(Graphic Structure)：该结构中的数据元素存在着多对多的关系，如图 1.1(d)所示。



(a) 集合　(b) 线性结构　(c) 树形结构　(d) 图状结构

图 1.1　类基本数据结构关系

由于集合中的元素的关系极为松散，可用其他数据结构来表示它，所以本书不做专门介绍。关于集合的概念将在 1.3.1 小节中介绍。

数据结构(Data Structure，DS)是一个二元组，其形式化定义为：

$$DS=(D,R)$$

式中：D 是数据元素的有限集合；R 是数据元素之间关系的有限集合。

下面通过例题来进一步理解后 3 类数据结构。

【例 1-1】学生信息表是一个线性的数据结构，如表 1.1 所示，表中的每一行是一个记录(在数据库管理系统中，表中的一个数据元素称为一个记录)，它由学号、姓名、所在班级、性别和出生年月等数据项组成。表中数据元素之间的关系是一对一的关系。

表 1.1　学生信息表

学　号	姓　名	所在班级	性　别	出生年月
040303001	雷洪	软件 04103	男	1986.12
040303002	李春	软件 04103	女	1987.3
040303003	周刚	软件 04103	男	1986.9

【例 1-2】家族关系是典型的树形结构，图 1.2 所示为一个三代的家族关系。图中爷爷、儿子、女儿、孙子、孙女或外孙女是一个结点(在树形结构中，数据元素称为结点)，他们之间是一对多的关系。其中，爷爷有两个儿子和一个女儿，这是一对三的关系；一个儿子有两个儿子(爷爷的孙子)，这是一对二的关系；另一个儿子有一个儿子(爷爷的孙子)和一个女儿(爷爷的孙女)，这也是一对二的关系，女儿有 3 个女儿(爷爷的外孙女)，这是一对三的关系。树形结构具有严格的层次关系，爷爷在树形结构的最上层，中间层是儿子和女儿，最下层是孙子、孙女和外孙女。不能倒过来，因为不会先有儿子或女儿再有爷爷，也不会先有孙子或孙女再有儿子等。

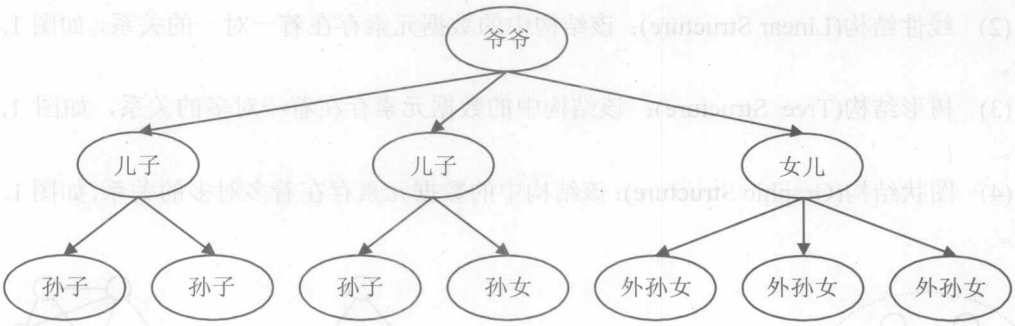


图 1.2　家族关系

【例 1-3】图 1.3 所示为 4 个城市的公路交通图，这是一个典型的图状结构。图中的每个城市是一个顶点(在图状结构中，数据元素称为顶点)，它们之间是多对多的关系。成都与都江堰、青城山、雅安直接通公路，都江堰与成都、青城山直接通公路，青城山与都江堰、成都及雅安直接通公路，雅安与成都、青城山直接通公路。这些公路构成了一个公路交通网，所以，又把图状结构称为网状结构。

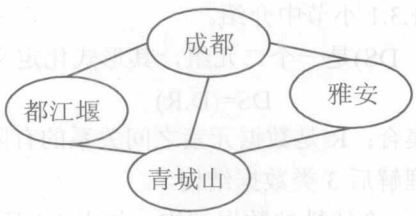


图 1.3　四城市交通图

从数据类型、抽象数据类型和数据结构的概念可知三者的关系非常密切。抽象数据类型包含一般数据类型的概念，但含义比一般数据类型更广、更抽象。一般数据类型由具体语言系统内部定义，直接提供给程序员定义用户数据(但对于结构类型，程序员要给出结构名和成员定义)，因此称它们为预定义类型。抽象数据类型通常由程序员定义，包括定义它所使用的数据和在这些数据上进行的操作。在定义抽象数据类型中的数据部分和操作部分时，要求只定义数据的本质规律和操作说明，不考虑数据是如何存储的和操作是如何实现的。这样抽象层次更高，更能为其他用户提供良好的使用接口。

数据类型可以看作是简单的数据结构。数据的取值范围可以看作是数据元素的有限集合，而对数据进行操作的集合可以看作是数据元素之间关系的集合。

对于抽象数据类型的描述，除了必须描述它的数据结构外，还必须描述定义在它上面的操作。抽象数据类型上定义的操作以该抽象数据类型的数据所应具有的数据结构为基础。本书把抽象数据类型看作是数据结构的一种表现形式，使用抽象数据类型来描述每一种数据结构。

数据结构包括数据的逻辑结构和物理结构。上述数据结构的定义就是数据的逻辑结构，数据的逻辑结构是从具体问题抽象出来的数学模型，是为了讨论问题的方便，与数据在计算机中的具体存储没有关系。然而，讨论数据结构的目的是为了在计算机中实现对它的操作，因此还需要研究在计算机中如何表示和存储数据结构，即数据的物理结构。数据的物

理结构又称为存储结构，是数据在计算机中的表示(又叫映像)和存储，包括数据元素的表示和存储以及数据元素之间关系的表示和存储。

数据的存储结构包括顺序存储结构和链式存储结构两种。顺序存储结构是通过数据元素在计算机存储器中的相对位置来表示出数据元素的逻辑关系，一般把逻辑上相邻的数据元素存储在物理位置相邻的存储单元中。在 C 和 C#语言中用数组来实现顺序存储结构，因为数组所分配的存储空间是连续的，所以数组天生就具有实现数据顺序存储结构的能力。链式存储结构对逻辑上相邻的数据元素不要求其存储位置必须相邻，链式存储结构中的数据元素称为结点，在结点中附设地址域来存储与该结点相邻的结点的地址来实现结点间的逻辑关系。这个地址在 C 语言中称为指针，这个地址域称为指针域；在 C#语言中称为引用，这个地址域称为引用域。

从数据结构的逻辑结构的概念可知，本书前面所说的“使用抽象数据类型来描述每一种数据结构”实际上是描述数据结构的逻辑结构。在数据结构的逻辑结构的 C 语言描述中，使用函数集合描述数据操作的集合；在 C#语言描述中，使用接口描述数据操作集合。关于接口请参阅 1.5.1 小节的内容介绍。

《数据结构》在 20 世纪中期之前不是一门独立的课程，其有关内容只是散见于其他课程之中，如编译原理及操作系统等课程。1968 年，美国的唐·欧·克努特教授开创了数据结构的最初体系。他所编著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统阐述数据的逻辑结构和存储结构及其操作的著作。从 20 世纪 60 年代末到 70 年代初，出现了大型程序，软件也相对独立，人们越来越重视数据结构，认为程序设计的实质是确定数据结构，加上设计一个好的算法，这就是人们常说的“程序=数据结构+算法”。下一节谈谈算法的问题。

1.2　算　　法

从上节可知，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，下面只从算法的特性、算法的评价标准和算法的时间复杂度三个方面进行介绍。

1.2.1　算法的特性

算法(Algorithm)是对某一特定类型的问题的求解步骤的一种描述，是指令的有限序列。其中的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(Finity)。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(Unambiguousness)。算法的每一个步骤都必须有确切的含义，即无二义性，并且对于相同的输入只能有相同的输出。

(3) 输入(Input)。一个算法具有零个或多个输入，即它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(Output)。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(Realizability)。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，但两者有区别。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。还有，一个程序只能用计算机语言来描述，也就是说，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。

本书尽可能采用 C 和 C#语言来描述和实现算法，使读者能够阅读或上机查阅，以便更好地理解算法。

1.2.2 算法的评价标准

对于一个特定的问题，采用的数据结构不同，其设计的算法一般也不同，即使在同一种数据结构下，也可以采用不同的算法。那么，对于解决同一问题的不同算法，选择哪一种算法比较合适，以及如何对现有的算法进行改进，从而设计出更适合于数据结构的算法，这就是算法评价的问题。评价一个算法优劣的主要标准如下。

(1) 正确性(Correctness)。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也是最基本的标准。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(Readability)。算法主要是为了人阅读和交流，其次才是机器的执行。所以，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理解。同时，一个可读性强的算法也有助于对算法中隐藏错误的排除和算法的移植。

(3) 健壮性(Robustness)。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，使得不至于引起严重的后果。健壮性要求表明算法要全面细致地考虑所有可能出现的边界情况，并对这些边界情况做出完备的处理，尽可能使算法没有意外的情况。

(4) 时间效率(Running Time)。时间效率是指算法在计算机上运行所花费的时间，它等于算法中每条语句执行时间的总和。对于同一个问题，如果有多个算法可供选择，应尽可能选择执行时间短的算法。一般来说，执行时间越短，性能越好。

(5) 空间存储量需求(Storage Space)。空间存储量需求是指算法在计算机存储上所占用的存储空间，包括存储算法本身所占用的存储空间、算法的输入及输出数据所占用的存储空间和算法运行过程中临时占用的存储空间。算法占用的存储空间是指算法执行过程中所需要的最大存储空间，对于一个问题如果有多个算法可供选择，应尽可能选择存储量需求低的算法。实际上，算法的时间效率和空间效率经常是一对矛盾，相互抵触。在时间运用中要根据问题的需要进行灵活处理，有时需要牺牲空间来换取时间，有时需要牺牲时间来换取空间。

通常把算法在运行过程中临时占用的存储空间的大小称为算法的空间复杂度(Space

Complexity)。算法的空间复杂度比较容易计算，它主要包括局部变量所占用的存储空间和系统为实现递归所使用的堆栈占用的存储空间。

如果算法是用计算机语言来描述的，还要看程序代码的大小。对于同一个问题，在用上面 5 条标准评价的结果相同的情况下，代码量越少越好。实际上，代码量越大，占用的存储空间会越多，程序的运行时间也可能越长，出错的可能性也越大，阅读起来也就越麻烦。

在以上标准中，本书主要考虑程序的运行时间，也考虑执行程序所占用的空间。影响程序运行时间的因素很多，包括算法本身、输入的数据以及运行程序的计算机系统等。计算机的性能由以下因素决定。

(1) 硬件条件。硬件条件一般包括所使用的处理器的类型和速度(比如，使用双核处理器还是单核处理器)、可使用的内存(缓存和 RAM)以及可使用的外存等。

(2) 实现算法所使用的计算机语言。实现算法的语言级别越高，其执行效率相对越低。

(3) 所使用的语言的编译器/解释器。一般而言，编译的执行效率高于解释，但解释具有更大的灵活性。

(4) 所使用的操作系统软件。操作系统的功能主要是管理计算机系统的软件和硬件资源，为计算机用户方便使用计算机提供一个接口。各种语言处理程序如编译程序、解释程序等和应用程序都在操作系统的控制下运行。

1.2.3　算法的时间复杂度

一个算法的时间复杂度(Time Complexity)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，算法中基本操作重复执行的频度 T(n)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，如 $T(n)=\frac{1}{2}n(n-1)$，则 $\frac{1}{2}n(n-1)$ 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 $O(n^3)$、对数阶 $O(\log_2 n)$等。

下面举例来说明计算算法时间复杂度的方法。

【例 1-4】分析以下程序的时间复杂度。

```
x = n;                  /*n>1*/
y = 0;
while(y < x)
{
```

```
    y = y + 1;                ①
}
```

解：这是一重循环的程序，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 1-5】分析以下程序的时间复杂度。

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

解：这是二重循环的程序，外层 for 循环的循环次数是 n，内层 for 循环的循环次数也为 n，所以，该程序段中语句①的频度为 n*n，则程序段的时间复杂度为 $T(n)=O(n^2)$。

【例 1-6】分析以下程序的时间复杂度。

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

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

【例 1-7】分析以下程序的时间复杂度。

```
for(i = 0; i < m; ++i)
{
    for(j = 0; j<t; ++j)
    {
        for(k = 0; k < n; ++k)
        {
            c[i][j] = c[i][j] + a[i][k] * b[k][j];          ①
        }
    }
}
```

解：这是三重循环的程序，最外层 for 循环的循环次数为 m，中间层 for 循环的循环次数为 t，最里层 for 循环的循环次数为 n，所以，该程序段中语句①的频度是 m*t*n，则程序段的时间复杂度是 T(n)=O(m*t*n)。