青少年信息学（计算机）奥林匹克竞赛培训教程系列丛书

全面 ★ 实用 ★ 权威

NOI

数据结构与程序实现

■司存瑞　苏秋萍　编著

西安电子科技大学出版社
http://www.xduph.com

青少年信息学(计算机)奥林匹克竞赛培训教程系列丛书

数据结构与程序实现

司存瑞　苏秋萍　编著

西安电子科技大学出版社
2009

内 容 简 介

青少年信息学(计算机)奥林匹克竞赛培训教程系列丛书是由从事青少年信息学奥林匹克竞赛教育多年、具有丰富竞赛辅导和教学经验的一线教师共同精心编著而成的。

《数据结构与程序实现》是这套丛书的第二册。全书共分 6 章，第 1 章介绍了数据结构与算法的概念，第 2 章至第 6 章分别讲述了线性表、栈和队列、数组、矩阵和串、树、图的基本概念，存储结构，基本运算与程序实现以及它们的应用。为了使学生尽快了解、掌握竞赛的内容和范围，我们特意从近年来国际、国内信息学奥林匹克竞赛中精选了若干试题，在各章(第 1 章除外)中增加了“典型试题分析”的内容。对这些试题，应用本章所讲内容完全可以解决。

本书深入浅出，思路清晰，不仅能帮助刚刚迈进信息学奥林匹克竞赛大门的选手掌握数据结构与算法的基本知识，更能从启迪思维、开发智力的角度引导他们如何使用计算机来分析问题和解决问题。

本书既可以作为全国信息学奥林匹克竞赛的培训教材和自学用书，也可以作为 ACM 大学生程序设计竞赛及大专院校相关专业教师和学生的参考教材。

为方便读者，我们对每章提供的典型算法、例题、习题均给出了参考程序，使用者可在西安电子科技大学出版社网站 http://www.xduph.com 下载。

★本书配有电子教案，有需要者可从出版社网站下载，免费提供。

图书在版编目(CIP)数据

数据结构与程序实现 / 司存瑞，苏秋萍编著. —西安：西安电子科技大学出版社，2009. 4

(青少年信息学(计算机)奥林匹克竞赛培训教程系列丛书)

ISBN 978 - 7 - 5606 - 2209 - 5

Ⅰ. 数… Ⅱ. ① 司… ② 苏… Ⅲ. 数据结构—中小学—教材 Ⅳ. G634.671

中国版本图书馆 CIP 数据核字(2009)第 020467 号

策　　划　霍小齐

责任编辑　王跃华　霍小齐

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　陕西华沐印刷科技有限责任公司

版　　次　2009 年 4 月第 1 版　2009 年 4 月第 1 次印刷

开　　本　787 毫米×1092 毫米　1/16　印　张　32

字　　数　765 千字

印　　数　1～4000 册

定　　价　48.00 元

ISBN 978 - 7 - 5606 - 2209 - 5/TP • 1125

XDUP 2501001-1

前　言

经教育部和中国科协批准，由中国计算机学会主办的全国青少年信息学(计算机)奥林匹克竞赛(National Olympiad in Informatics，简称 NOI)，是一项全国性的青少年学科竞赛活动。与此相关的活动还有：国际信息学奥林匹克竞赛(International Olympiad in Informatics，简称 IOI)和全国青少年信息学(计算机)奥林匹克联赛(National Olympiad in Informatics in Provinces，简称 NOIP 或全国联赛)。它们都是为普及我国青少年的计算机知识而举行的活动。

信息学奥林匹克竞赛系列活动的宗旨就是在青少年中普及计算机科学，通过组织信息学奥林匹克竞赛可使学生开阔眼界、扩大知识面，了解计算机在现代化社会中的战略地位，培养并促其能力得以发展；通过竞赛发现人才，对有才华的青少年起到激励作用。这些活动给学校信息技术课程增加了动力并启发了新的思路，对全国计算机普及教育的兴盛及教学内容的改革也一直起着良性的导向和指导作用。

信息学奥林匹克竞赛是智力与应用计算机能力的比赛，对选手除了要求其具有扎实的基础知识、掌握计算机的程序设计语言、熟悉数据结构与算法外，还需要有较强的上机编程、调试程序的能力，特别是算法的设计与上机调试程序的能力。这就需要竞赛选手在有限的时间内，用自己所掌握的知识与具备的能力，凭借可能的工具，通过编写程序，有效地解决实际问题。

解决实际问题的过程一般应当遵循下面的流程：

理解问题并建立相应的数学模型

↓

根据数学模型的特点选用适当的数据结构及算法

↓

将算法用程序实现并调试正确

但是，在实际中往往会看到这样的情况：一部分选手在比赛时找出了正确的算法，但由于时间不足而造成程序未能完成，其结果与不会做的选手一样；另一部分选手，他们想到的算法大体一致，但在程序实现时有优劣之分，测试(在时间与空间方面)的结果也就相差甚远了。

为什么会出现这样的情况呢？一般来讲，这是由于选手没有熟练、扎实、深入的基本功所致。具体体现在平时对编程环境不够了解，运用不够熟练，忽视了将算法转变为程序实现这一环节的重要性和必要性。

在解决实际问题的过程中，我们强调算法是首要的，具有战略性的地位。但是一个算法再好，如果不能转化为正确的程序，那么，对解决这一实际问题来说，其效果就是零。这就是算法用程序实现的必要性。与此同时，算法实现的快慢，决定了选手能否有时间去

解决其他问题；算法实现的好坏，决定了算法的表现，这主要体现在对程序的测试过程中。在有多人想到了正确算法的情况下，其算法的高质量程序实现就显得尤为重要了。这就是算法用程序实现的重要性。

算法与其程序实现的关系，是战略与战术的关系，算法起决定性作用，从根本上决定了算法质量的优劣，但算法的程序实现反过来也影响着算法，并且在一定的条件下，这种影响超过了算法本身的作用。为此，我们聘请具有丰富教学和竞赛辅导经验的一线教师共同编写了青少年信息学(计算机)奥林匹克竞赛培训教程系列丛书，其中，第一册为《程序设计与基本算法》，第二册为《数据结构与程序实现》，第三册为《算法设计与分析技巧》，试图从语言基础、数据结构和算法分析的高度来训练青少年程序设计的技能。

本书是这套丛书的第二册。目前，国内外“数据结构”方面的教科书，大都是从理论上对算法加以阐述，程序设计部分以伪代码的形式给出，而学生和用户迫切需要了解的是“这种算法用程序如何实现”。本书作者从事“数据结构”课程教学多年，对这门课程本身有比较深刻的理解，对学生们学习该课程的需要和难点所在有着比较清楚的认识。实际上，只有真正掌握了数据结构中各种算法的程序实现过程，所学知识才能在解决实际问题中发挥作用。这一问题的解决就是需要研究者有较为扎实的理论基础，同时要有丰富的编程实践能力。根据教学经验，我们在本书的内容安排和知识点的处理方面作了一些尝试，力求体现以下特点：

1．对每一种数据结构的基本运算以软芯片的形式给出，使读者看到，每种运算只不过是芯片上的一只管脚而已。当用户需要执行某种运算时，只需接上相应的管脚即可。

2．在叙述数据结构的内容时，穿插了算法设计和思路分析，并在程序实现部分给出了完整代码。

3．文字简练明了，对难点剖析详尽，重点算法和典型问题的分析均给予注释，以方便读者彻底弄懂弄通。

4．删繁就简，突出数据结构的核心内容。事实上，数据结构所涉及的内容还有一些，如查找、排序等，由于这些内容在《程序设计与基本算法》一书中已有提及，这里略去。

5．为了使读者尽快掌握信息学奥林匹克竞赛的内容和范围，除第 1 章外，其余各章均从近年来各级各类信息学奥林匹克竞赛中精选了若干试题，增加了“典型试题分析”一节内容。这些试题，应用相应章所讲内容完全可以解决。另外，我们对每章提供的典型算法、例题、习题均给出了参考程序，读者可在西安电子科技大学出版社网站 http://www.xduph.com 下载。

本书由司存瑞、苏秋萍共同编著。张晨、司栋、黄希敏等同志参加了部分章节的编写与程序调试，他们为本书的出版付出了辛勤的劳动，在此表示感谢。

尽管作者讲授“数据结构”课程并从事青少年信息学奥林匹克竞赛教育多年，但由于时间仓促，不妥之处在所难免，恳请读者批评指正。

作　者

2008 年 10 月

目　录

第 1 章　数据结构与算法的概念

自从 1946 年世界上第一台计算机诞生以来，计算机科学与技术的飞速发展和广泛应用远远超出了人们的预料。如今，计算机已渗透到现代社会的各行各业及方方面面，并在一定程度上改变了人类的活动方式和思维习惯。与此同时，计算机处理的对象也从单纯的数值数据发展到各种不同形式的数据，如字符、表格、声音、图像等。与纯粹的数值数据不同，非数值数据往往带有一定的“结构”，即数据之间的关联，它是数据所代表的客观对象之间相互联系的对应物。计算机在存储与处理这类数据时，也必须同时存储和处理这些“结构”，只有这样才能在计算机中完整地再现客观世界，进而实现用计算机来进行控制或管理。如何合理、有效地组织和存储这些信息，以支持高速的计算机信息处理，这是计算机科学研究的重点之一。研究各种数据及其相互之间的关系、研究如何有效地存储数据、研究如何设计出性能良好的数据处理程序等，这些都是“数据结构”课程所要讨论的主要内容。因此，数据结构与算法不仅是计算机科学与技术学科的基础，也是计算机科学与技术学科永恒的主题。

1.1　数据结构的概念

所谓数据结构，简单来说就是数据及其之间的关系，其中，数据间的不同关系种类决定了不同种类的数据结构。本书将介绍四种基本数据结构：线性表、栈和队列、树与图。

在计算机科学中，研究一种数据结构主要从以下四个方面来进行：

(1) 数据结构的逻辑结构——特定的数据结构的定义；

(2) 数据结构的物理结构——在计算机中存储这种数据结构的方法；

(3) 该数据结构上常用的操作方法；

(4) 该数据结构的应用。

在许多情况下，也把上述四个方面中的前三个方面合称为数据结构，而在另外一些场合下，数据结构又单指其逻辑结构或物理结构，所以名词“数据结构”的确切含义取决于其上下文。

本书将从第 2 章开始，从上述四个方面讨论常用的数据结构。本章是一个简单的前导，主要讨论各种数据结构的共性问题，介绍研究数据结构的方法、工具以及其他相关问题。

1.1.1　数据、数据元素与数据类型

计算机中的数据在计算机内部是原始的一组组二进制代码，程序设计语言以这种代码为基础建立起了所有的数据。数据的概念不再只是那些用数字组合而成的各种数据了，如

整数、小数、实数、复数、指数和对数等。随着计算机科学技术的发展，数据的概念也相应地发生了一些重要的变化。

数据(Data)是对客观事物的符号表示，在计算机科学中是指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。这是计算机程序加工的“原料”。例如利用数值分析方法解代数方程的程序，其处理对象是实数；编译程序或文字处理程序所处理的对象是字符串等。因此，对计算机科学而言，数据的含义极为广泛，声音、图像等都可以通过编码而纳入数据的范畴。

数据的基本单位是数据元素(Data Element)，有时也称作元素、结点、顶点、记录等。一个数据元素也可以由若干个数据项(Data Item)组成。数据项是具有独立含义的数据不可再分的最小标识单位。例如，一个单位的职工花名册中，每一位职工的信息就是一个数据元素，职工信息中含有职工编号、姓名、性别、出生日期、参加工作时间、政治面貌、职称、职务、工资级别等项目，每一个项目都是某个职工数据元素中的一个数据项。

数据、数据元素、数据项反映了数据组织的 3 个层次，即数据可以由若干个数据元素组成，数据元素又可以由若干个数据项组成。数据项是对数据元素属性的描述，数据元素是对客观世界中某个独立个体的数据描述。在 C 语言和 Pascal 中，数据元素是分别用结构体和记录来描述的，每个数据项则是结构体或记录中的一个分量。

计算机中的数据可以按类型来划分，划分的结果就是数据对象。所谓数据对象(Data Object)，是指具有相同性质的数据元素的集合，是数据的一个子集。如整数数据是集合 N={0，±1，±2，…}，字母字符数据对象是集合 C={'A'，'B'，'C'，…}等。程序设计语言大都提供有一种简单的数据对象，如整数、实数、布尔值、字符值等，并提供了以此为基础来构造新数据对象的机制。在一个具体问题中，数据元素具有相同性质，属于同一数据对象，数据元素是数据对象的一个实例。如在前述的职工花名册中，所有的职工是一个数据对象，不同的职工是不同的数据元素，他们都是职工数据对象的不同实例，其数据元素值是各数据项的一个具体描述。

数据类型(Data Type)是对在计算机中表示的同一数据对象及其在该数据对象上的一组操作的总称。如整数数据，在计算机中它是集合{minint，…，maxint}上的整数(其中 minint 和 maxint 分别是最小整数和最大整数，在不同的计算机中表示的值不同)，且这个集合是有限集合，是数学意义上无穷集合的一个子集；在这个集合上可以进行的操作有加、减、乘、整除和求模等算术运算以及等于、不等于、大于、小于、大于等于和小于等于等关系运算。数据对象整数以及在整数集合上算术运算和关系运算等操作一起构成了整数这个整数类型。

数据类型的概念是程序设计语言和程序设计过程中一个非常重要的概念，它具有以下六个显著的特征：

① 类型决定了变量或表达式所有可能取值的全体成员集合。如整型决定了整型变量或整型表达式所有取值在计算机可以表示的整数范围(16 位微机为–32 768～+32 767)；字符型决定了字符变量或字符表达式所有取值为计算机可以表示的所有字符集合(一般为 ASCII 码集界定的符号集合)；而布尔型则决定了布尔型变量或表达式的所有取值为真(True/非零)或(False/零)。

② 每一个值隶属于且仅隶属于某一类型。如值 3 隶属于整型而 3.0 隶属于实型，值'A'隶属于字符型而 True 隶属于布尔型。

③ 任何常量、变量或表达式的类型，都可以从其形式上或所处的上下文关系中推断出来，无需了解在程序运行时计算出来的具体值。其中数值可以从形式上看出类型，如 3 和 3.0 分别属于整型和实型；标识符常量和变量可以通过前面的类型定义看出其所属类型；而表达式则可以由构成表达式中的成分常量、变量或函数的类型推断出来。

④ 每一种操作都要求一定类型的操作数据，且得出一定类型的操作结果。如关系运算操作要求两个操作数应是同一类型的有序类型数据，结果的类型为布尔型；而实数运算则要求两个操作数类型相容(实型或整型)，结果为实型，等等。

⑤ 一种类型的值及其在该类型上规定的基本操作的性质可由一组公理来阐明，如整型值的有序性和整型运算操作加和乘的交换律、结合律以及乘对加的分配律等。

⑥ 高级程序设计语言使用类型信息去防止程序中出现无意义的结构，又由类型信息确定在计算机中的数据表示和数据处理方法。不同类型的数据在计算机中的表示方法不同，如字符型用 1 个字节，整型用 2 个字节(长整型用 4 个字节)，而实型也是用 2 个字节或 4 个字节，但又分为指数和尾数两个部分来表示。不同类型的数据对象可施加的运算操作不同，相应的处理方法也不同。编译程序(或解释程序)利用类型信息检查操作和动作的合法性，如除法操作“/”仅限于实数值，而整除操作仅限于整数值，赋值操作仅当赋值号左边的变量类型与赋值号右边的表达式类型相同或相容时才有意义等。类型检查防止程序中出现无意义的结构，从而改善了对程序错误的检查。

1.1.2　数据结构的概念

数据结构(Data Structure)是指计算机程序中所操作的对象——数据以及数据元素之间的相互关系和运算。在任何问题中，数据元素之间都不会是独立的，总是存在着这样或那样的关系，这种数据之间的关系也称作结构。数据结构包括以下 3 个方面的内容：

① 数据的逻辑结构即数据元素之间的逻辑关系。它只抽象地反映数据元素集合的结构，而不管其存储方式，可用一个二元组给出如下的形式定义：

$$\text{Data Structure}=(D, R)$$

其中，D 是数据元素的集合，R 是 D 上的关系的集合。从结构的观点出发，一般可将数据结构分为两大类，即线性结构和非线性结构。线性结构有线性表、栈、队列、串、数组和文件等；非线性结构有树、图和集合等。

② 数据的存储结构即数据及数据元素之间的关系在计算机内存中的表示，也称作数据的物理结构或数据映像。主要的存储方式有顺序存储和链式存储两种，此外，还有索引存储和散列存储等其他方式。从逻辑结构到存储结构称为映像。同一逻辑结构采用了不同的存储结构存储，就会得到不同的数据结构。这是因为映像变了，使结构有了变化，使得实现逻辑结构上所定义的运算的算法也随之改变了。

③ 数据的运算及实现。程序中的数据运算是定义在数据的逻辑结构上的运算，但运算的实现要在相应的存储结构上进行。常用的运算有检索、插入、删除、更新、排序等。在数据的逻辑结构上定义数据运算时，只考虑这些运算是“做什么”，而不考虑它“如何做”；只有在选定了某种数据结构的存储结构时，才去考虑如何具体实现这些运算(即运算的实现)。运算的实现依赖于所选取的存储结构，也依赖于所选用的程序设计语言。在本书中讨论的算法，均以 Free Pascal 语言实现。

在线性结构中，D 中数据元素之间存在着一对一的次序关系，如图 1.1(a)所示。其逻辑特征为：存在一个唯一被称做“第一个”的数据元素，它没有前趋只有一个直接后继(Immediate Successor)；存在一个唯一被称做“最后一个”的数据元素，它没有后继只有一个直接前趋(Immediate Predecessor)；其他数据元素都有且仅有一个直接前趋，也有且仅有一个直接后继。如职工花名册、学生成绩表、向量、数组、购物时排的队等都是线性结构的例子。第一个数据元素有时也称做开始结点，最后一个数据元素有时也称做终端结点。

在非线性结构中，集合 D 中的数据元素之间不存在一对一的次序关系。树形结构中的数据元素之间，存在着一对多的层次关系，如图 1.1(b)所示。在树形结构中，没有直接前趋的结点称为根结点，除根结点外每个结点有且仅有一个直接前趋(称为双亲结点)；没有直接后继的结点称为叶结点，除叶结点外每个结点都有一个或多个直接后继(称为孩子结点)。树的例子很多，如族谱中的家族树，计算机文件管理中的目录树，编译程序中用到语法树等；就连我们所用的教材也是由书、章、节、小节、知识点构成的多层次结构的树。

非线性结构中的图状结构，其数据元素之间既不存在线性结构中的一对一次序关系，也不存在树形结构中的一对多层次关系。在图结构中，D 中数据元素之间的关系是多对多的网状关系，如图 1.1(c)所示。换句话说，图是一种网状结构，任意两个数据元素之间都可能相关；其中的每一个数据元素，既可以有多个直接前趋，也可以有多个直接后继。交通网络图，课程之间的先修后修的关系图，软件开发过程中所有用到的程序图、控制流图、数据流图等都是图结构的例子。

非线性结构中的集合结构，其 D 中数据元素之间关系是“属于同一个集合”，如图 1.1(d)所示。集合是数据元素关系极为松散的一种结构，通常是用其他结构来表示集合，不专门讨论它的实现方法。

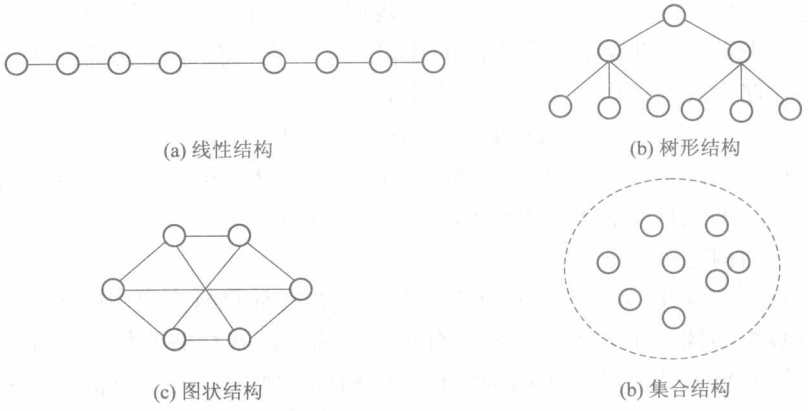

图 1.1　数据结构示意图

把数据元素及其数据元素之间的关系在计算机内表示出来是数据存储结构研究的问题。下面介绍几种常用的存储表示方式，如顺序存储、链式存储、索引存储、散列存储等。

顺序存储方式就是在计算机内存储器中开辟一片地址连续的存储单元顺序存放数据中的各个元素，它把逻辑上相邻的数据元素存放在物理上相邻的存储单元中，利用物理上的邻接关系表示逻辑上的先后次序关系，这种存储表示方式称做顺序存储结构(Sequential Storage Structure)。顺序存储结构是一种最基本的存储方法，通常借助于程序设计语言中的

数组来实现，主要用于线性数据结构的存储，对于非线性结构进行线性化处理后也可实现顺序存储。

链式存储方式是把数据元素和反映元素间关系(后继和/或前趋)的地址一块存储在计算机内，它不要求在内存储器中开辟的存储单元地址连续，数据元素可以存放在内存储器中的任意位置，借助指示数据元素存储的指针表示元素间的逻辑关系，这种存储表示方式称作链式存储结构(Linked Storage Structure)。链式存储结构也是一种基本的存储表示方法，通常借助于程序设计语言中的指针来实现。主要用于树形结构和图结构数据的存储，为了某种特殊的需要也常用于一些线性结构的存储。

索引存储方式和散列存储方式，通常是为了检索方便所采用的存储表示方法(此内容以后会详细介绍，在此从略)。

一般来说，这几种基本的存储表示方法，既可以单独使用，也可以结合起来使用。选择何种存储结构要依具体问题的要求而定，既要考虑问题表示和运算的方便性，也要考虑到实现算法的时间和空间效率要求。

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都取决于所选定的数据的逻辑结构，而算法的实现则依赖于所采用的存储结构。各种数据结构分别提供了不同类型的数据在作为计算机程序数据时的组织、管理、存储、运算和处理的相应方法和技术，其中有些数据结构在程序设计语言中已经实现了，并作为数据类型提供给程序设计人员使用，如各种基本类型；也有些数据结构已经在程序设计语言中实现了，并提供了程序设计人员在自己的程序中定义数据类型的方法或手段，如数组、字符串等用户自定义类型；还有许多数据结构，在程序设计语言中没有实现，要靠程序设计人员利用语言中提供的某些设施去实现或模拟实现，如栈、队列、树、二叉树、图、网络等。在程序设计语言中实现了的数据结构称为数据类型，本书中讨论的侧重点是那些在程序设计语言中尚未实现的数据结构。

1.1.3　抽象数据类型

抽象的本质就是抽取反映问题本质的东西，忽略掉非本质的细节。数据类型是对同一数据对象及其在该数据对象上的一组操作的抽象。抽象的结果是把用户不必了解的一切细节都封装在类型之中，对使用数据类型的用户来说是实现了信息隐藏(Information Hiding)。用户在使用数据类型时，既不需要了解该数据类型在计算机内部是如何表示的，也不需要知道其操作是如何实现的，只需集中精力考虑所要求解的问题应该如何去解决。抽象数据类型的概念，是我们熟知的数据类型概念的引申和发展，是数据抽象和运算抽象紧密结合的产物。

抽象数据类型(Abstract Data Type，ADT)是指一个数据模型以及定义在该数据模型上的一组操作。这里的数据模型，是指要求解问题中的各种数据的总和。通常，可以把这些数据组成为一个或多个数据结构。当数据模型表现为一个数据结构时，抽象数据类型就是这个数据结构以及定义在这个数据结构上的一组运算。这种情况是我们讨论和学习抽象数据类型概念的基础，也是数据结构课程对抽象数据类型定义的根本要求。当数据模型表现为多个数据结构时，可以先定义以单个数据结构为基础的各个抽象数据类型，然后在此基础上(需要的话)定义抽象级别更高的抽象数据类型，并利用它们构造最终的问题求解算法。

抽象数据类型的定义仅取决于它的一组逻辑特性，而与它在计算机内部如何表示和实现无关。也就是说，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；抽象数据类型可以通过已有的数据类型来表示和实现，利用已有的数据类型来说明新的结构，利用已经实现了的操作来完成新的操作。下面以大家都熟知的自然数为例，给出抽象数据类型自然数的 ADT 描述：

```
ADT natural number
{Data Objects：D={x | x∈natural number，0≤x≤maxint}
    /*定义自然数的数据对象为 0 到计算机可以表示的最大整数*/
Data Relation：R={<x-1,x> | x∈natural number ,1≤x≤maxint}
    /*说明自然数中元素之间的关系是一对一的次序关系 */
Elementary Operation:
 /* x,y∈natural number；False，True∈boolean；+、−、<、==、=等均可服务*/
Zero( )
返回 0
Iszero(x)
 if(x==0)  返回 Ture else 返回 False
Add(x,y)
 if(x+y<=maxint)    返回   x+y   else 返回 maxint
Equal(x,y)
  if(x==y)  返回 Ture else 返回 False
    Successor(x)
 if(x==maxint)    返回 x   else 返回 x+1
Subtract(x,y)
 if(x<y)  返回 0   else 返回 x−y
}end ADT natural number;
```

抽象数据类型是算法设计和程序设计中的重要概念，这个概念明确地把数据结构与作用在该结构上的运算紧密地联系起来。数据结构上的运算依赖于其具体表示，有了数据结构的具体表示和运算的具体实现，运算的效率随之确定。需要指出的是，基本数据类型已隐含了数据结构和定义在该结构上的运算。且每一种基本数据类型都连带着一组基本运算，只是由于这些基本数据类型中数据结构的具体表示和基本运算的具体实现都很规范，都通过内置而隐藏起来使得人们看不到它们，故人们习惯于在算法与程序设计中使用基本数据类型名和相关的运算而不问其究竟。

由于实际问题千奇百怪，数据结构千姿百态，问题求解算法千变万化，抽象数据类型的设计和实现不可能像基本数据类型那样规范。首先要根据问题的具体要求，定义该抽象数据类型需要包含哪些信息，并根据功能确定公共界面的服务，使用者可以使用公共界面中的服务对该抽象数据类型进行操作。另一方面，把抽象数据类型的实现作为私有部分封装在其实现模块内，让使用者看不到、也不能直接操作该类型所存储的数据，只能通过界面中的服务来访问这些数据。这样做的好处是严格区分了抽象数据类型的两个视图。从使用者的角度来看，只需了解该抽象数据类型的规格说明，就可以利用其公共界面中的服务

使用它，而不必关心其具体实现，有利于在开发过程中抓住重点集中精力解决要求解的问题。从实现者的角度来看，把抽象数据类型的具体实现封装起来，有利于编码和测试，也有利于将来的修改。这样做可以使得错误局部化，一旦出现错误其传播范围不至于影响其他模块；如果为了提高效率改进数据结构，可能需要改变抽象数据类型的具体实现，但只要公共界面中服务的使用方式不变，其他所有使用该抽象数据类型的程序都可以不变，从而大大提高了系统的稳定性和整体效率。

1.2　算法的概念和描述

Anany Levitin 在他的名著《算法设计与分析基础》第 1 章绪论中一开篇引用了 David Berlinski 对算法的评价：

科技殿堂里陈列着两颗熠熠生辉的宝石，一颗是微积分，另一颗就是算法。微积分以及在微积分基础上建立起来的数学分析体系成就了现代科学，而算法则成就了现代世界。

算法，对于计算机专业人士，无论从理论还是从实践的角度来说，对其学习和研究都是必需的。这是因为，从实践的角度来看，必须了解计算领域中不同问题的一系列标准算法，此外还要具备设计新的算法和分析其效率的能力。从理论的角度来看，对算法的研究已被公认为是计算机科学的基石。

对于非计算机的相关人士，学习算法的理由也是非常充分的，坦率地说，没有算法，计算机程序将不复存在，更不用说使用它了。而且，随着计算机日益渗透到我们的工作和生活的方方面面，需要学习算法的人也越来越多。

1.2.1　算法的概念

虽然对算法的概念没有一个大家公认的定义，但我们对它的含义还是有基本共识的。

算法是一系列解决问题的清晰指令，也就是说，对于符合一定规范的输入在有限步骤内求解某一问题所使用的一组定义明确的规则。通俗点说，就是计算机解题的过程。在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。

这个定义可以用一幅简明的图来说明(如图 1.2 所示)。

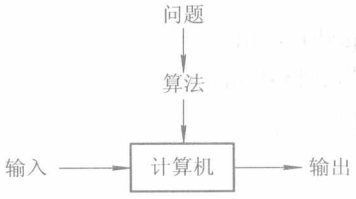

图 1.2　算法的概念

一个算法应该具有以下五个重要的特征：

① 有穷性：一个算法必须保证执行有限步之后结束，并且每一步都在有穷时间内完成。

② 确定性：算法的每一步骤必须有确切的定义。

③ 输入：一个算法有零个或多个输入，以刻画运算对象的初始情况。

④ 输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

⑤ 可行性：算法应该是可行的，这意味着所有有待实现的运算都能够由相应的指令所指示的操作来执行，并可通过有限次运算完成。

为了阐明算法的概念，本节将以 3 种方法为例来解决同一个问题：计算两个整数的最大公约数。这些例子会帮助我们阐明几项要点：

- 算法的每一个步骤都必须没有歧义，不能有半点含糊。
- 必须认真确定算法所处理的输入的值域。
- 同一算法可以用几种不同的形式来描述。
- 同一问题，可能存在几种不同的算法。
- 针对同一问题的算法可能会基于完全不同的解题思路而且解题速度也会有显著不同。

还记得最大公约数的定义吗？两个不全为 0 的非负数 m 和 n 的最大公约数记为 gcd(m，n)，代表能够整除(即余数为 0)m 和 n 的最大正整数。亚历山大的欧几里得(公元前 3 世纪)所著的《几何原本》，以系统论述几何学而著称，在其中的一卷里，他简要描述了一个最大公约数算法。用现代数学的术语来表述，欧几里得算法基于的方法是重复应用下列等式，直到 m mod n 等于 0。

gcd(m，n) = gcd(n，m mod n)　　(m mod n 表示 m 除以 n 之后的余数)

因为 gcd(m，0) = m(为什么？)，m 最后的取值也就是 m 和 n 的初值的最大公约数。

举例来说，gcd(60，24)可以这样计算：

gcd(60，24) = gcd(24，12) = gcd(12，0) = 12

下面是该算法的一个更加结构化的描述：

用于计算 gcd(m，n)的欧几里得算法

第一步：如果 n = 0，返回 m 的值作为结果，同时过程结束；否则，进入第二步。

第二步：m 除以 n，将余数赋给 r。

第三步：将 n 的值赋给 m，将 r 的值赋给 n，返回第一步。

我们也可以使用伪代码来描述这个算法：

```
算法 Euclid(m，n)
//使用欧几里得算法计算 gcd(m，n)
//输入：两个不全为 0 的非负整数 m，n
//输出：m，n 的最大公约数
while n ≠ 0 do
r←m mod n
m←n
n←r
return m
```

上面的伪代码也可以用流程图来加以描述：

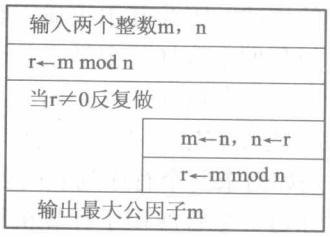

图 1.3　欧几里得算法的流程图

我们怎么知道欧几里得算法最终一定会结束呢？通过观察我们发现，每经过一次循环，参加运算的两个算子中的后一个都会变得更小，而且绝对不会变成负数。确实，下一次循环时，n 的新值是 m mod n，这个值总是比 n 小。所以，第二个算子的值最终会变成 0，此时，这个算法也就停止了。

就像其他许多问题一样，最大公约数问题也有多种算法。让我们看看解这个问题的另外两种方法。第一种方法只基于最大公约数的定义：m 和 n 的最大公约数就是能够同时整除它们的最大正整数。显然，这样一个公约数不会大于两数中的较小者，因此，我们先有：t= min{m，n}。我们现在可以开始检查 t 是否能够整除 m 和 n：如果能，t 就是最大公约数；如果不能，我们就将 t 减 1，然后继续尝试(我们如何确定该算法最终一定会结束呢？)。例如，对于 60 和 24 这两个数来说，该算法会先尝试 24，然后是 23，这样一直尝试到 12，算法就结束了。

用于计算 gcd(m，n)的连续整数检测算法

第一步：将 min{m，n}的值赋给 t。

第二步：m 除以 t，如果余数为 0，进入第三步；否则，进入第四步。

第三步：n 除以 t，如果余数为 0，返回 t 的值作为结果；否则，进入第四步。

第四步：把 t 的值减 1。返回第二步。

注意，和欧几里得算法不同，按照这个算法的当前形式，当它的一个输入为 0 时，计算出来的结果是错误的。这个例子说明了为什么必须认真、清晰地规定算法输入的值域。

求最大公约数的第三种过程，我们应该从中学起就很熟悉了。

中学里计算 gcd(m，n)的过程

第一步：找到 m 的所有质因数。

第二步：找到 n 的所有质因数。

第三步：从第一步和第二步求得的质因数分解式中找出所有的公因数(如果 p 是一个公因数，而且在 m 和 n 的质因数分解式分别出现过 p_m 和 p_n 次，那么应该将 p 重复 $\min\{p_m, p_n\}$ 次)。

第四步：将第三步中找到的质因数相乘，其结果作为给定数值的最大公约数。

这样，对于 60 和 24 这两个数，我们得到：

$60 = 2 \times 2 \times 3 \times 5$

$24 = 2 \times 2 \times 2 \times 3$

$\gcd(60, 24) = 2 \times 2 \times 3 = 12$

虽然学习这个方法的那段中学时光是令人怀念的，但我们仍然注意到，第三个过程比欧几里得算法要复杂得多，也慢得多。撇开低劣的性能不谈，以这种形式表述的中学求解