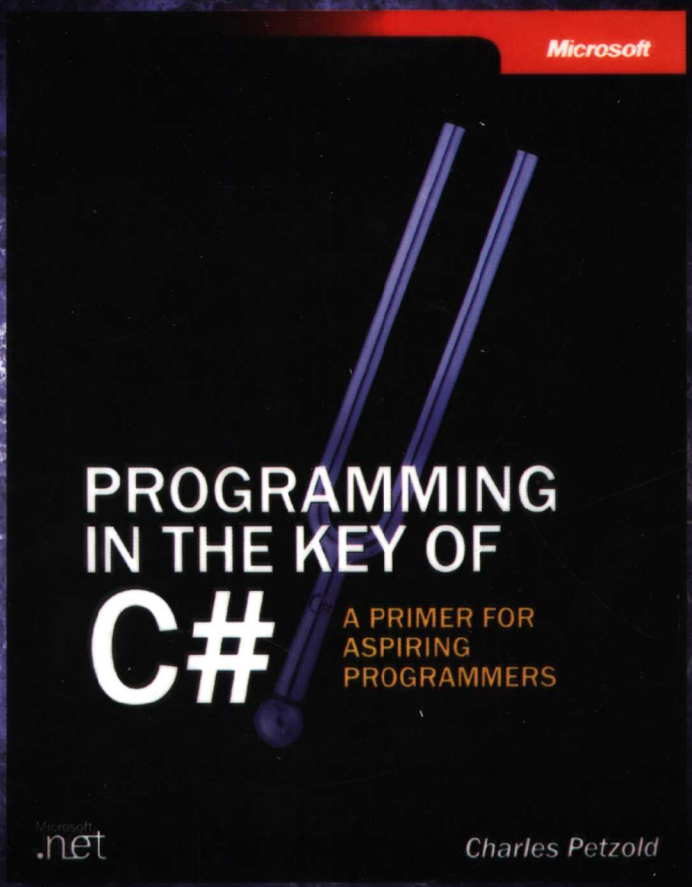
Microsoft 计算机科学丛书

C#程序设计

(美) Charles Petzold 著　杨涛　王建桥　杨晓云　高文雅　译



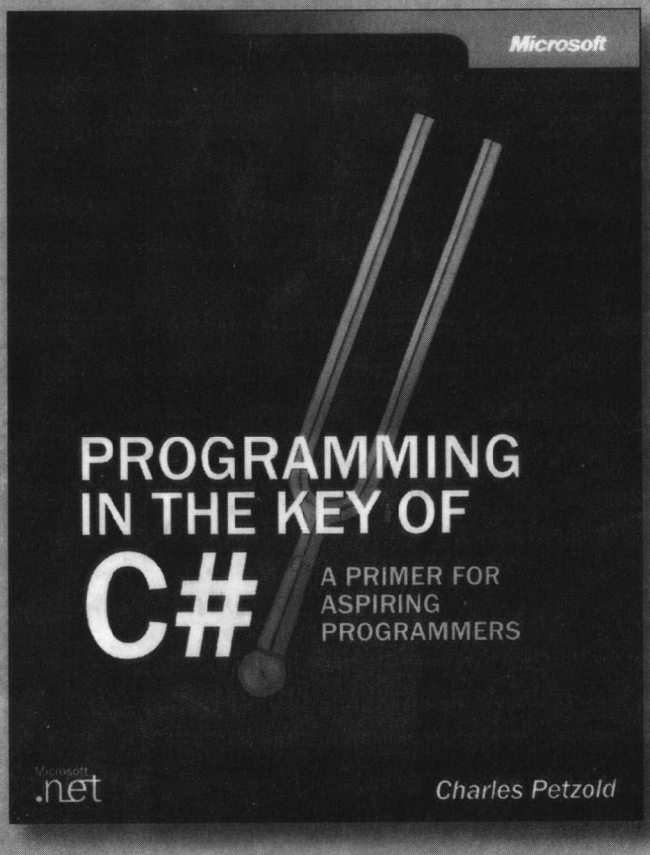
Programming in the Key of C#

机械工业出版社
China Machine Press

计 算 机 科 学 丛

C#程序设计

(美) Charles Petzold 著　　杨涛　王建桥　杨晓云　高文雅　译



Programming in the Key of C#

机械工业出版社
China Machine Press

由获奖作家Charles Petzold撰写的这本书对C#语言做了深入浅出、循序渐进的论述。不论是第一次接触程序设计还是第一次接触C#语言，你都会迅速掌握使用C#语言开发应用程序的技巧。

本书版权登记号：图字：01-2004-1649

图书在版编目（CIP）数据

C#程序设计 /（美）佩佐尔特（Petzold, C.）著；杨涛等译. - 北京：机械工业出版社，2004.4

（计算机科学丛书）

书名原文：Programming in the Key of C#

ISBN 7-111-13988-7

Ⅰ. C… Ⅱ. ① 佩… ② 杨… Ⅲ. C语言 - 程序设计 - 高等学校 - 教材 Ⅳ. TP312

中国版本图书馆CIP数据核字（2004）第009819号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：于杰琼

北京昌平奔腾印刷厂印刷·新华书店北京发行所发行

2004年4月第1版第1次印刷

787mm × 1092mm 1/16 ·19.75印张

印数：0 001-5 000册

定价：30.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

前　言

本书是为那些希望学习C#程序设计语言的人们编写的。本书尽量避免对读者此前的程序设计经验做出任何假设。本书将从最基本的变量开始讲起，最后以一个实现演奏音乐功能的程序结束。

C#是微软公司开发的一种面向对象的现代程序设计语言。这种语言是微软公司在2000年夏季发布的.NET战略的一个组成部分。读者可以用C#和.NET编写Web应用或者在Microsoft Windows环境下运行的程序。

C#并不是进行.NET程序设计时惟一可用的程序设计语言。微软公司提出的CLS（Common Language Specification，通用语言标准）对能被用来编写.NET程序的语言的最小功能集做出了规定。许多现有的程序设计语言都符合CLS的有关规定，但C#是专为.NET而设计和开发的程序设计语言，它最接近.NET战略的功能要求。

正如其名称所暗示的那样，C#是各种C和C++语言的后代，它与同样受到C和C++语言很深影响的Java语言有很多相似之处。人们把由C语言衍生出来的这些语言统称为“C家族”语言，它们有着相同或近似的语法，但在具体实现细节上却有着很大的不同。就拿C#来说，它是目前程序设计语言“以效率换安全”发展潮流的一个产物。正如本书第1章讨论的那样，C和C++之所以如此流行，部分原因是因为用它们编写出来的程序运行速度都非常快，占用的内存也最少。但这些语言是在“程序员都非常聪明、不会犯任何错误”的假设下才获得如此之高的效率的；它们本身并没有提供任何能够检查程序行为是否正确（比如程序是否会去访问不允许它去访问的内存等）的检查机制。C和C++程序可能执行得非常快，但代码里通常会存在一些很难诊断的漏洞。

如今的计算机速度越来越快，内存也越来越便宜，所以程序本身的执行效率已不再是人们关注的焦点——人们如今更关心程序中的漏洞是否尽可能地少。虽说没有一种程序设计语言能够让程序员编写出完全没有漏洞的程序，但与C和C++相比，C#在这方面已经有了巨大的进步。因此，C#程序运行起来更安全。

C#程序的安全性还体现在其他方面。对于经由网络传播的程序，用户需要确定它们不会对自己的计算机和数据造成任何破坏。C#（以及Java）在这方面是有保障的。传统上程序员使用的基本工具之一是能把源代码（程序员写出来的代码）转换为可执行代码（由计算机去执行的代码）的编译器。但C#编译器却把源代码转换为一个包含有中间语言（intermediate language，简称IL）代码的文件，而这个文件是不能在计算机上直接执行的。IL必须被转换为一个可执行文件才能执行，而这一步骤将由.NET中的CLR组件（Common Language Runtime，通用语言运行库）负责完成。

本人在2000年初次接触C#语言，但它很快就成为我最喜欢用的程序设计语言。C#沿袭了C语言语法的简洁性——这种简洁性是我从1985年开始学习C语言起就一直非常欣赏的，但摒弃了C语言中那些用处不大的东西并增加了许多现代的、面向对象的特性。C#语言是各种新、旧程序设计语言的一种近乎完美的组合。

在开始使用C#不久，我就意识到它是一种非常适合新程序员学习的入门级程序设计语言。C#既精致又强大，能帮助新程序员避免犯很多常见的错误。这促使我下决心编写本书。

前面讲过，C#非常适合用来编写.NET Web应用程序或基于Windows的应用程序，但这不过是些用户操作界面而已，不属于本书的讨论重点。这本书是关于C#程序设计语言本身的。因此，在编写本书时，我尽量把讨论重点放在C#语言本身并希望不会让大家有所分心，但今后打算用C#来做些什么却完全取决于读者自己。当然了，程序总是要与外部世界进行某种形式的通信。在本书中，将使用“控制台”来作为有关程序的用户操作界面。控制台的好处是它本身非常简单，但它涉及的知识却适用于任何形式的程序。比如说，在学完本书之后，如果你想编写一些基于Windows的应用程序，可以直接跳到我的另一本书《Programming Microsoft Windows with C#》（Microsoft 出版公司2001年出版）中的第2章。

系统要求

读者可以用很多种软件开发工具来编写和编译C#程序，这类工具会越来越多。我已尽量使本书独立于各种开发工具，但在某些不得不涉及C#程序创建过程的示例里，将重点介绍两种办法。

用C#编写程序的标准做法是购买一套Microsoft Visual C# .NET Standard Edition（其零售价大概是100美元）或Microsoft Visual Studio .NET Professional Edition软件（其零售价大概是1000美元）。Visual Studio .NET还包含有对C++、Microsoft Visual Basic .NET以及其他一些高级功能的支持。在本书中，在不需要区分这两种开发工具的场合下，将使用Visual C# .NET来指称它们。要想运行Visual C# .NET，需要运行Windows 2000或Windows XP，而硬件配置至少应该满足以下要求：Pentium III 600 MHz、160MB内存、2GB硬盘可用空间（Visual Studio .NET需要4.2 GB）、1024 × 768 × 256色的显示能力。

如果使用的是2002版本的Visual C# .NET（而不是2003或更高版本的Visual C# .NET或者任何版本的Visual Studio .NET），就必须再增加一个小文件才能创建空白项目。这方面的具体细节可以在我的Web站点http://www.charlespetzold.com上的“Programming in the Key of C#”页面里查到。

我不知道Visual C# .NET是否是最适合初学者使用的开发工具。它体积庞大，功能复杂，有很多初学者根本用不上的东西。因此，我特意为大家准备了一个相对小一点的程序，我把这个程序叫作“Key of C#”，读者可以从我的Web站点免费下载它。要想使用“Key of C#”，首先要下载和安装.NET Framework Software Development Kit（SDK），你们可以在http://msdn.microsoft.com/downloads/list/netdevframework.asp处找到它（任何版本的.NET Framework SDK都能配合本书使用；需要运行Windows 2000、Windows XP或更高版本的Windows；下载文件的大小大约是100MB），再到http://www.charlespetzold.com站点上的“Programming in the Key of C#”页面里找到并安装“Key of C#”就行了。

如果你已经安装了Visual C# .NET，在安装和使用“Key of C#”之前就用不着再下载和安装.NET Framework SDK了。这个SDK里的所有东西都已经包括在Visual C# .NET里。

如果读者喜欢在Windows下的MS-DOS命令行上进行工作并希望能够继续这样做，也可以在下载并安装了.NET Framework SDK（或Visual C# .NET）之后打开一个MS-DOS窗口来进行编程。在我的Web站点上的“Programming in the Key of C#”页面里有很多关于如何使用

C#命令行编译器来配合本书学习的信息。

支持

本书中的示例程序可以从Microsoft出版公司的Web站点http://www.microsoft.com/mspress/books /6261.asp处下载：点击该页面右边“More Information”（更多信息）菜单里的“Companion Content”（相关内容）链接，打开“Companion Content”页面后，就可以从中选择想要下载的示例文件（在我的Web站点http://www.charlespetzold.com上的“Programming in the Key of C#”页面里也有一个指向Microsoft出版公司Web站点的链接）。示例程序文件包括解决方案文件（.sln）、适用于Visual C# .NET环境的C#项目文件（.csproj）以及适用于我的“Key of C#”程序的“Key of C#”项目文件（.kcsproj）。

我们已尽了最大努力来保证本书及有关源代码内容的准确性。Microsoft出版公司通过下面这个网址为其出版的各种书籍提供了勘误表：

http://www.microsoft.com/mspress/support

如果想直接进入Microsoft出版公司的知识库并查询某个具体的问题，请访问下面这个站点：

http://www.microsoft.com/mspress/support/search.asp

如果有与本书有关的评论、疑问或者建议，可以通过以下两条途径发送给Microsoft出版公司：

普通邮件：
Microsoft Press
Attn: Programming in the Key of C#
One Microsoft Way
Redmond, WA 98052-6399

电子邮件：mspinput@microsoft.com

注意，上面这个电子邮件地址不提供对有关软件产品的支持。如果有Visual C# .NET、Visual Studio .NET或.NET Framework等方面的问题，请访问微软公司的产品支持站点：

http://support.microsoft.com

至于与“Key of C#”程序有关的支持，请访问我的Web站点：

http://www.charlespetzold.com

虽然没有微软公司那么多的资源，但我将尽量支持好“Key of C#”程序。

致谢

如果没有我妻子Deirdre的关爱和支持、没有我们共同建立起来的家庭所营造出来的舒适与宁静，我是不可能写出这本书来的。

这里还要提到我与朋友们在星期天、星期二和星期四的聚会，那些聚会给了我很多帮助和支持。

我是在我的经纪人Moore版权代理公司的Claudette Moore和Microsoft出版公司的高级编辑Danielle Voeller的共同鼓励下开始写这本书的。他们并没有给我规定严格的交稿期限，这使我能够按照自己的想法去写作本书。我的好友Bruce Eckel充分肯定了我选择控制台I/O作为用户操作界面的做法，而以后的事实也证明了这是惟一适当的选择。

比与一位Microsoft出版公司的项目编辑和一位技术编辑共同合作更好的事情是什么？是与两位项目编辑和两位技术编辑共同合作！Denise Bankaitis和Jim Fuchs做了大部分工作，但因为暑假和其他日程安排方面的原因，Sally Stickney和Julie Xiao也加入了进来。如果没有这几位编辑细心阅读我的手稿，这本书里肯定会有许多不正确的论述和漏洞百出的代码。编辑们不仅纠正了冗余的代码和多余的文字论述，更重要的是他们帮助作者从多个角度去看待和分析问题。我还要特别感谢我的良师益友Robin A. Reynolds-Haertle，他找出了许多不正确的假设、错误的逻辑论述和各种画蛇添足的文字。对于读者仍能在这本书里发现的问题，我将承担全部责任。

我的职业生涯受Wendy Carlos的影响颇深，他在1968年写的《Switched-On Bach》一书不仅使一位16岁的少年喜欢上了巴赫的音乐，还让他迷上了电子音乐。十年之后，我开始自学数字电子技术、制作我自己的由计算机控制的电子乐器，并逐渐相信自己在电子谱曲以外还有与芯片和代码打交道的其他天分。我希望本书最后一节里的程序能够表达我对Wendy Carlos和他的《Switched-On Bach》一书的敬意。

最后，我要把这本书奉献给Johann Sebastian Bach本人——少数几位有勇气用C大调谱写音乐的作曲家之一。

Charles Petzold

纽约

2003年6月

目　录

第1章 基 础

深夜，当结束了一天的工作但大脑仍很活跃的时候，程序员往往会问自己这样一个问题：程序设计到底是一门艺术还是一门科学？

它看起来两个都是。作为一名艺术家，程序员以基本的软件开发工具作为画笔，以系统中的可用内存作为画布，用自己的激情从零星的字节创造出一个东西来。但这块画布并不允许程序员随心所欲地涂抹，蕴涵在程序设计工具中的语义规则允许你创造出各种各样的东西，但它们又像自然规律那样严格和不可违反。

在现今时代，程序员既是设计师又是建设者，既是建筑师又是建筑工人，既是有远见卓识的领袖又是埋头苦干的工程师。我们建造的村落将成为未来的国际化都市，把更多的人、更多的团体和更多的信息有机地链接在一起并形成一个不断扩大的范围，而这种创造之美只有那些真正有过这种体会的人才能欣赏。那些不是程序员的人很难欣赏程序员们的创造工作——当程序工作正常时，他们不知赞扬；但当程序出问题时，他们却会大加指责；当老程序被新东西所替代时，他们就会喜新厌旧——而这一切几乎已经成为用户们的习惯了。

一个计算机程序就像是一架壮观的机器，当这架机器里的齿轮、杠杆和活塞运转起来的时候，整个房间就将充满着华丽复杂而又美妙动听的音乐。我们将看到变化着的逻辑、运动着的算法和舞蹈着的数据。这是一幅不同寻常的画面，但确实是程序员眼中所见：一段段的程序以物质世界无法匹敌的精确度在相互合作着。在我看来，生活中很少有比眼看着一个新程序突然激活并投入运行更让人激动的事情了。

大多数计算机用户永远不会体会到这种喜悦。如今的计算机用户有很多根本不是程序员，也永远不会成为程序员；他们或者是简单地运行着其他人编写出来的各种应用程序（文字处理软件、电子邮件程序、Web浏览器等），或者是一无所知地使用着内嵌有各种计算机程序的电子设备（移动电话、DVD机、烤面包机等）。

其实事情并不总是如此。在大约1970年以前，想不编写程序就弄出点有用的东西几乎是不可能的。计算机革命其实发生了两次——第一次发生在计算机刚被设计和制造出来的时候，第二次发生在几十年后当不是程序员的人们也能使用计算机的时候。我认为，自从计算机“用户”出现之后，程序员和用户就再也无法像以前那样沟通和交流了。

第一台有实用价值的数字化可编程计算机诞生于20世纪30年代。这里所说的“数字化”指的是机器在工作中使用的是诸如0、1、3.1415、1.86×10^6之类的真实数字，“可编程”指的是计算机能够根据人们事先编写好的代码指令（即所谓的“程序”）完成一系列复杂的数学运算。有很长时间，计算机程序都是些穿在纸带或卡片或其他介质上的小孔；人们把计算机本身称为“硬件”，把编码指令称为“软件”。

“软件”得名于它很容易被改变，你用不着重新制造一台计算机就能进行另一种计算。“硬件”则被设计成能执行一系列基本的数学运算和逻辑操作。硬件将执行哪些操作以及将以何种顺序来执行那些操作都由程序代码来控制。“计算机执行一个程序”的意思就是读入有关指令代码并执行它。

在数字化计算的前几十年里，硬件与软件的结合十分紧密。每台机器都有着独一无二的指令集。如果你想把一个程序拿到另一台更新更快的机器上去运行，就必须把那个程序改写为全新的代码。这些依赖于具体机器的指令集被称为“机器代码”或者（当人们开始使用英文符号来代表这些代码的时候）“汇编语言”。

从20世纪50年代早期开始，各种计算机语言逐渐被设计出来，它们使程序设计工作不再依赖于某具体计算机的体系结构和指令集。人们把这些语言称为“高级语言”，把原来的机器代码称为“低级语言”。有些早期的高级语言至今仍有人在使用。比如说，FORTRAN语言（FORmula TRANslation，意思是“公式翻译”）在科学家和工程师当中仍有很多的追随者，COBOL语言（Common Business Oriented Language，意思是“面向商业用途的通用语言”）在金融领域的大型主机中也仍广泛地使用着。于20世纪60年代中期诞生的BASIC语言（Beginner's All-purpose Symbolic Instruction Code，意思是“供初学者使用的通用性符号化指令代码”）也仍很流行，但如今的BASIC与其早期版本相比已经有了相当大的差异。

用FORTRAN、COBOL、BASIC或者其他高级语言编写的程序基本上都是些文本文件，其内容是严格按照相关语言所定义的语法写出来的一系列语句。因为计算机只能执行机器代码，所以我们必须用一个“编译器”或者一个“解释器”程序把用高级语言写出来的程序转换为机器代码。“编译器”把整个程序一次性地全部转换为将被计算机执行的机器代码；“解释器”则是转换一条语句、执行一条语句，如此交替进行直到程序语句全部处理完毕。

在20世纪50年代晚期还出现了一种如今已不再有人使用（就我个人所知）、但对程序设计语言的发展有着重大影响的程序设计语言——ALGOL（ALGOrithmic Language，意思是“算法语言”，但也有人说它得名于天文学中英仙座第二亮的星星）。ALGOL是由一个国际组织于20世纪50年代后期设计的，该组织还在1960年和1968年对ALGOL进行了两次改进和修订。虽说ALGOL如今已成为一种“死”语言，但它的精神却仍存在于它的后继者如Pascal、PL/I、C等语言中。

C语言诞生于贝尔实验室。贝尔实验室对现代世界的影响可以说是无处不在。在1947年，贝尔实验室发明了晶体管；在20世纪70年代，UNIX操作系统也诞生于此。多年以来，与UNIX操作系统结合最紧密的程序设计语言就是C，它基本上是由Dennis Ritchie设计的。有很多人对C语言这个名称的由来感到好奇：C语言得名于一种早期的程序设计语言B，而B语言又是BCPL（Basic CPL）语言的一个简化版本，BCPL又是从CPL（Combined Programming Language，混合式程序设计语言）衍生出来的。

与其他程序设计语言相比，C语言要紧凑得多。C语言以左、右花括号作为语句块的首尾标志，ALGOL（以及由它派生出来的许多语言）使用的则是关键字“BEGIN”和“END”。用C语言编写出来的程序通常都有着很高的效率，与其他程序设计语言相比，用C程序编译出来的机器代码在数量上一般要少很多。C语言最具吸引力的特点之一是它的“指针”类型；各种指针——简单地说，指针就是内存中的地址——使C程序能够非常方便地完成很多工作。因为C语言能够直接对比特、字节和内存进行处理，所以有不少人把C语言称为“高级汇编语言”。

C语言的高效率是有代价的，任何一位使用过C语言的程序员都知道它有多么的危险。大多数程序设计语言的编译器都会在程序里额外插入一些机器代码指令以防止程序做“坏”事或者发生崩溃。但为了使生成的代码运行速度最快，C编译器不会在程序里额外插入这类机器

代码指令。C语言假设程序员总是比编译器聪明，对程序的检查不像其他语言那么严格，所以即使是很有经验的C程序员也很容易写出有问题的代码来。虽说“程序漏洞”是程序设计工作中不可避免的副产品，但C编译器却经常会对一些无法在其他程序设计语言的编译器中蒙混过关的错误视而不见。这些错误中的大多数都与指针有关，即允许程序往不应该写入的内存区域写入数据。

C语言仍是一种很流行的程序设计语言，但现在的它已经显得有点过时了。根据C程序的结构特点，人们把它归类为一种传统的“过程化语言”：一个C程序通常由多个“过程”（或者叫“函数”或“子例程”）组成，而一个“过程”就是一段能够完成某种任务或者实现某种算法的代码，这些“过程”将对“数据”——即数字、文本、或者数字和文本的组合——进行各种各样的处理。在一个传统的过程化程序设计语言里，数据将由代码进行处理。

而如今的程序员更喜欢“面向对象”的程序设计语言。面向对象的程序设计技术（object-oriented programming，OOP）起源于一种名为SmallTalk的程序设计语言，这种语言是由帕洛阿尔托研究中心（Palo Alto Research Center，PARC）研发的。PARC是施乐公司（Xerox）创办的一家研究中心，Apple Macintosh和Microsoft Windows操作系统中很多图形化用户界面的原始概念都是由该中心最早提出来的。

在面向对象的程序设计语言中，程序员将创建出各种各样的“类”（class）而不是“过程”，再用这些类衍生出“对象”（object），一个“对象”就是由代码和数据构成的一个混合体。数据不再由“过程”进行处理，数据本身就带有对自己进行处理的工具。也就是说，数据将由它本身进行处理。这种观念上的转变大大提高了代码的“复用性”，使程序员能够把同一段代码反复运用到多种程序任务中。

曾经有很多勇于进取的程序员试图为C语言创建一个面向对象的版本，其中最为成功且流行开来的是贝尔实验室的Bjarne Stroustrup于20世纪80年代早期开发的C++。C++得名于C语言中的“++”操作符，这个操作符将对它的操作数加1（参见1.5节）。

C++有一些天生的不足。从理论上讲，C++是C语言的一个超集，它没有替换C语言中的任何功能，但增加了一些新东西。这迫使C++不得不引入一些被人们认为难看和难用的记号方式。

不仅如此，曾被认为是C和C++中最有特色的功能——用指针进行底层处理的能力——越来越被认为是它的一个先天不足。计算机技术在最近几年里的迅猛发展使人们不再把性能和经济性作为评判程序优劣的基本标准。硬件变得越来越快，内存变得越来越便宜，这些以前最为稀缺的资源如今已不再成为人们关心的焦点。现在的人们更希望代码里的漏洞尽可能地少，哪怕为此牺牲一些C语言的效率也不足惜。

到了20世纪90年代，Sun Microsystems公司推出了Java，这种面向对象的程序设计语言同样以C语言为蓝本，但与C++却有着显著的区别。Java摒弃了C++某些难看又难用的语法，也剔除了某些有危险性的C语言功能，但它保留了C语言简洁紧凑的特点。

到了2000年，微软公司发布了由Anders Hejlsberg主持开发的C#（念作“C sharp”），这个语言同样延续了C系列语言的命名传统。“#”字符看起来像是把“C++”中的两个加号合并在了一起。此外，在音乐里，C#（C大调）的音阶要比C高一点儿。

类似于Java，C#也摒弃了C语言里的一些危险功能。虽然C#并没有完全禁止使用指针，但大多数程序设计工作都可以在不使用指针的情况下完成。

C#与Java的另一个相似之处是编译器的角色。传统意义上的编译器将把“源代码”（用高级语言写出来的文本文件）转换为机器代码。机器代码形成一个“可执行文件”，这个文件可以直接在计算机上运行。但因为机器代码是与某特定计算机相关联的，所以可执行文件只能在特定类型的计算机上执行。这正是不能把为Apple Macintosh开发的程序直接拿到Microsoft Windows上运行的原因。

C#编译器将把源代码转换为一种“中间语言”（intermediate language，简称IL），这是一种通用性的机器代码。只有当在某特定计算机上运行程序时，IL才会被转换为真正的机器代码。这种转换对用户来说是透明的。从理论上讲，这种两个步骤的过程将使同一个IL程序在不同类型的计算机上都能运行。此外，IL形式的程序也使得操作系统更容易发现其中的恶意性或破坏性指令，而这种能力对于那些通过因特网传播的程序有着非常重要的意义。

人们把C、C++、Java和C#统称为“C语言家族”或者“基于C的语言”。C++是C语言的一个超集，它不仅具备C语言的一切功能，还增加了一些东西。严格地讲，Java和C#并不是C语言的超集，它们之间的相似之处要比它们与C语言的相似之处更多。但这几种程序设计语言在语法方面有很多其他程序设计语言所没有的相通之处。在这本书里，为了让大家对C、C++和C#的演变情况有一个了解，会经常提到C#的某个功能是否继承自C语言、它与C语言有何区别、或者它在C语言里是否根本就不存在。

人们经常把学习一种程序设计语言与学习一门外语相提并论，它们也的确有相似之处。比如说，它们都有一个词汇表——虽然程序设计语言的词汇表相对要小得多。C#只定义了77个保留字，我把它们附在本书的最后。出于保持C语言家族传统的考虑，C#的保留字全都为小写字母。

程序设计语言都有一套语法规则，但与人类语言相比，这些规则要简单但严格得多。这是因为计算机程序的语法必须足够简单才能保证编译器会对它做出完整且无二义性的解释。目前还没有什么程序能够同样轻松和同样精确地阅读人类语言。

学习一种程序设计语言和学习一门外语的最大的相似之处是不可能仅仅通过阅读一本书就熟练地掌握它。你必须进行实践，必须像学习演奏一种乐器那样不断地去实践。你必须亲自动手去编写你自己的代码。当你第一次编写代码时（这往往不是一个愉快的经历），就会知道你正在学着像一位程序员那样去思考了。

程序设计是一种解决问题的活动。程序员最重要的素质是具备把问题分解成一系列小环节的能力。C#所提供的工具只能解决这些小的环节，程序员的职责就是把问题的各个环节拼凑起来并拿出一个完整的解决方案。这个能力也需要不断的实践才能提高。

对细节的了解肯定会对你有所帮助。程序员不能容忍松散的思维或者只能在99%的时间里发挥功效的解决方案。通过这本书，我将尽量使大家能够像一个合格的程序员那样去思维——我也只能做到这么多了。类似于很多其他的程序设计教程，这本书里的很多示例程序都非常简短，它们只是用来说明C#程序设计工作中的某个特定概念或者技巧。要想编写出大程序，必须从这些简短的程序开始在亲身实践中去逐步地学习和积累。

在这本书里，我将尽量把C#示例程序与这种语言的实际应用环境区分开来。比如说，你可以用C#去编写可以在Windows下独立运行的程序，也可以把C#用在Web页面上。随着时间的推移，C#肯定会被人们用来编写其他类型的应用程序。各种各样的C#应用程序都需要通过各种方法从用户那里获得输入，再把输出显示给用户。C#并不要求你必须使用某种特定的输

入/输出（I/O）模型，C#应用程序的I/O模型不在本书的讨论范围之内。

为避免不必要的麻烦，本书将向大家演示如何为“控制台”（console）这种非常老式的I/O设备编写C#程序。在几十年前，程序员使用电传打字机和穿孔卡片来进行输入输出。后来，早期的个人电脑把显示器当作控制台——在Windows之类的图形化程序界面出现之前，用户就是通过显示器这张“脸”与MS-DOS等操作系统打交道的。即使在Windows下，控制台仍以Command Prompt（命令提示符）窗口的形式存在着。

虽说大家今后要编写的程序很可能不需要以控制台作为I/O设备，但学习控制台编程技术决不会是一种毫无用处的练习。在开发难度较大的代码或者在稍后的调试工作中，很多有经验的程序员都会使用控制台来简化问题。我可以向你们保证你在这本书里学到的东西都会派上用场。

像这样的一个教程可能既不完整又不精确。比如说，在下一节里，我写下了这样的话：“允许出现在等号左边的操作数类型只能是变量”。这句话并不完全正确。属性和数组类（数组、列表等）元素也可以出现在那里。但你在学到1.1节时还不需要知道那么多——我将在以后的章节里把它补充完整，你们最终将有一个完整的了解。

C#的正式文档叫作《C# Language Specification》，这份文档可以在Microsoft出版公司2001年出版的《C# Language Specifications》（注意，最后一个单词多了一个“s”）一书里查到，也可以在C#的发行版本里或者在网上找到这份文档。我希望大家能够利用空余时间好好读读这本书，但在开始阅读这本书之前，你需要对C#有足够的熟悉。在这本书里，我偶尔会引用一下《C# Language Specification》文档里的内容。这份文档的所有章节都方便地被编号为大纲格式。

虽说《C# Language Specification》是解答C#程序设计问题最权威的文档，但更友好的用户指南却是《C# Programmer's Reference》。你可以在C#发行版本、网上或者Microsoft出版公司出版的《Microsoft Visual C# .NET Language Reference》（2002）一书里查到它。它是一本非常有用的参考书，也收录了一些在别的地方查不到的信息。

这两份文档——以及更多的编程资料——都可以在MSDN（Microsoft Developer Network）网站（http: //msdn.microsoft.com）上查到。在我写这本书的时候，C#语言的网页是http: //msdn.microsoft.com/library /en-us/cscon/html/vcoriCStarPage.asp。如果你在那里找不到它，在我的Web站点上的“Programming in the Key of C#”页面上应该有一个最新的链接，该页面的地址是http: //www.charlespetzold.com/key。

1.1 变量

计算机和程序设计都是人们为了进行数值计算而发明出来的，所以很多程序设计语言都沿用了数学符号（包括用字母或单词来表示数字）的做法并不让人感到吃惊。

下面是几个简单的代数式：

```
A = 3
B = 2
C = A + B
```

这当然不是一个很难的问题，但如果我们能用计算机解决这个问题，就应该能用计算机解决一些更复杂的问题。首先，要把这些代数式转换为C#代码，如下所示：

```
A = 3;
B = 2;
C = A + B;
```

这三行代码并不能构成一个完整的C#程序，但你在一个典型的C#程序（比如你本人编写出来的某个C#程序）里很容易看到类似的代码。正如你将在1.3节里做的那样，你需要在一个“编辑器”程序（editor，它就像是一个简化了的文字处理软件）中敲入这些代码，再运行编译器程序把这些代码转换为一个可以拿到计算机上去执行的.EXE文件。

代数式与C#代码之间的明显区别只有一个：每行C#代码都以一个分号结尾。在C#里，这三行代码中的每一行都被称为一条“语句”。计算机程序就是由各种各样的语句构成的。字母A、B、C被称为“变量”；从技术角度讲，“变量”就是以符号形式给出的、该变量的值在内存中的存放位置。

代数式与C#代码之间这种表面上的相似其实有着本质的区别，这个区别体现在等号（=）的角色和作用上。等号在代码和在代数式中有着不同的含义。比如说，如果别人把以下三个代数式拿给你看：

```
A = 3
C = A + B
B = 2
```

你可能会犹豫一下，但最终仍会说“C仍等于5”。这是因为代数式“B = 2”所表达的是“B等于2”这个不可更改的事实，就像你说“喵喵是个猫”一样——它不可能再变成狗。

但如果像下面这样把刚才把那几条C#语句的顺序颠倒一下，情况就大不一样了：

```
A = 3;
C = A + B;
B = 2;
```

C#程序里的语句是按其先后顺序由计算机一条一条地执行的。第一条语句的效果是把变量A的值设置为3，但只有当计算机执行完这条语句后A才会等于3。接下来，当执行到第二条语句的时候，变量B的值还没有被设置成等于2——我们不知道B等于多少；我们看到的只是某个大程序中的几条语句而已。

在C#里，等号的正确名称是“赋值操作符”。一个“操作符”就是一个将使计算机执行某一个特定操作的符号或者单词。操作符有“操作数”。赋值操作符有两个操作数：出现在等号左边的一个变量和出现在等号右边的一个表达式。赋值操作符的作用是把等号右边的表达式的计算结果赋值给等号左边的变量。以上三条语句都是赋值语句的例子，而赋值语句是计算机程序中一种极为常见的语句类型。

赋值语句有很多种不同的写法。下面这些赋值语句在C#里都是合法的：

```
C = A + 3;
C = 27 + A + B;
C = B + A + A + B + A + B + 14;
C = 45 + 27;
```

但不管你怎么写，等号左边的变量总是被赋值为等号右边的表达式的计算结果。加号（+）是C#的“算术操作符”之一。当然，C#还允许进行减法、乘法和除法计算；这些将在1.5节进行讨论。