万水网络与安全技术丛书

HACKING
THE ART OF EXPLOITATION

黑客之道：漏洞发掘的艺术

（原书第二版）

[美] Jon Erickson 著 范书义 田玉敏 等译

随书赠送LiveCD，提供完备调试环境，让您即刻体验黑客技术的精髓

中国水利水电出版社
www.waterpub.com.cn

万水网络与安全技术丛书

黑客之道：漏洞发掘的艺术

（原书第二版）

[美] Jon Erickson 著

范书义 田玉敏 等译

中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

作为一本黑客方面的畅销书，本书完全从程序开发的角度讲述黑客技术，虽然篇幅不长，但内容丰富，涉及了缓冲区、堆、栈溢出、格式化字符串的编写等编程知识，网络嗅探、端口扫描、拒绝服务攻击等网络知识，以及信息论、密码破译、各种加密方法等密码学方面的知识。

通过阅读本书，读者可以了解黑客攻击的精髓、各种黑客技术的作用原理，甚至利用并欣赏各种黑客技术，使自己的网络系统的安全性更高，软件稳定性更好，问题解决方案更有创造性。

值得一提的是书中的代码示例都是在基于 x86 运行 Linux 的计算机上完成的，而本书附赠的 LiveCD 提供了已配置好的 Linux 环境，鼓励读者在拥有类似结构的计算机上一起进行实践。读者将看到自己杰作的结果，并不断实验和尝试新的技术，而这正是黑客所崇尚的精神。本书适合具有一定编程基础且对黑客技术感兴趣的读者阅读。

图书在版编目（CIP）数据

黑客之道：漏洞发掘的艺术 /（美）埃里克森（Erickson,J.）著；范书义等译.—北京：中国水利水电出版社，2009

（万水网络与安全技术丛书）

书名原文: Hacking: The Art of Exploitation

ISBN 978-7-5084-6620-0

Ⅰ. 黑… Ⅱ. ①埃…②范… Ⅲ. 计算机网络－安全技术 Ⅳ. TP393.08

中国版本图书馆 CIP 数据核字（2009）第 113131 号

策划编辑：杨庆川　责任编辑：李　炎　加工编辑：徐　雯　封面设计：无极书装

书　名	万水网络与安全技术丛书 **黑客之道：漏洞发掘的艺术（原书第二版）**
作　者	[美] Jon Erickson 著　范书义　田玉敏　等译
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路 1 号 D 座　100038） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）68367658（营销中心）、82562819（万水）
经　售	全国各地新华书店和相关出版物销售网点
排　版	北京万水电子信息有限公司
印　刷	北京市天竺颖华印刷厂
规　格	184mm×260mm　16 开本　27.25 印张　631 千字
版　次	2009 年 7 月第 1 版　2009 年 7 月第 1 次印刷
印　数	0001—4000 册
定　价	58.00 元（赠 1CD）

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

译者序

近年来，随着社会网络化、信息化程度的提高，网络和信息的安全越来越重要，“黑客”一词的出现频率也越来越高，但黑客的本质到底是什么？

“黑客”是一个外来词，是 hacker 的中文翻译。其实它本身没有什么特殊的含义，原意是指一些热衷于计算机和网络技术研究的人。这些人为计算机和网络世界发狂，对任何有趣的问题都会进行研究，他们的精神是一般人所无法领悟的。无可非议，这样的“黑客”是一个褒义词。但英雄谁都愿意做，慢慢地有些人打着黑客的旗帜做了很多并不光彩的事。黑客们称他们为骇客（creaker 或 cracker），并以他们为耻，不愿同他们做朋友。其实，黑客和骇客并没有十分明显的界限。他们都入侵网络，破解密码，但是他们的出发点上却有着本质的差别：黑客是为了网络安全而入侵，为了提高自己的技术而入侵。自由是黑客们的理想，他们梦想的网络世界是一个没有利益冲突，没有金钱交易，完全共享的自由世界。骇客却是为了一己私欲而进入到他人的系统大肆破坏。因此，黑客们拼命地研究，目的是为了完善网络，使网络更安全，而他们所进行的研究工作也多少带有一些神秘色彩。

本书讲解各种黑客攻击技术的细节，技巧性非常强。尽管本书着重介绍构造这些黑客攻击技术的程序设计基本概念，但一般的程序设计知识也的确能够帮助读者理解这些概念。本书中的代码示例都是在基于 x86 运行 Linux 的计算机上完成的。我们鼓励读者在拥有类似结构的计算机上一起进行实践，你将看到自己杰作的结果，并实验和尝试新的技术，而这正是黑客所崇尚的精神。

本书由范书义、田玉敏主译，张波、谢君英、盛海艳和吴爱金等也参与了本书的翻译和校对工作，在此一并表示感谢。由于译者水平有限和时间仓促，书中难免有疏漏和错误之处，敬请读者在阅读过程中不吝指正。

译者

2009 年 4 月

关于本书

黑客攻击是创造性的问题求解艺术，可以用来寻找某一难题的非常规解决方案或者发掘脆弱程序中的漏洞。许多人称他们自己为黑客，但很少有人具备作为一名成功的黑客必须具备的坚实技术基础。

本书的作者 Jon Erickson 并没有简单地介绍如何运行现有的漏洞发掘成果，而是讲解了神秘黑客技术的工作机制。为了使用一种人人都能理解的方式分享黑客艺术和黑客科学，本书从一个黑客的视角介绍了 C 语言程序设计基础。

随书附带的 LiveCD 提供了一个完整的 Linux 程序设计和调试环境，无需您修改自己当前的操作系统。在您弥补自己的知识缺口并独立探索黑客技术时，可以使用 LiveCD 和书中的示例来亲自动手调试代码、制造缓冲区溢出、劫持网络通信、绕过防护、发掘密码的弱点，甚至是发明一种新的漏洞发掘技术。本书将为您讲授：

- 使用 C 语言、汇编语言和 shell 脚本在计算机上编程
- 使用缓冲区溢出和格式化字符串破坏系统内存以运行任意代码
- 使用调试工具检查处理器寄存器和系统内存来真正理解到底发生了什么
- 智取类似于“不可执行堆栈”和“入侵检测系统”等的常见安全措施
- 使用“端口绑定”或反向连接 shellcode 获取远程服务器的访问权，并修改服务器的登录行为以实现隐身
- 重定向网络流量，隐匿开放端口，劫持 TCP 连接
- 使用 FMS 攻击来破解加密的无线通信，使用密码概率矩阵加快穷举暴力破解的速度

黑客们始终在打破界限、研究未知并不断发展他们的艺术。即使您还不知道如何编程，本书也将会为您展示程序设计、机器体系机构、网络通信和现存黑客攻击技术的全貌。将这些知识和随书附带的 Linux 环境结合起来，剩下的就看您的创造力了。

关于作者

Jon Erickson 受过计算机科学的正规教育，从 5 岁起就开始尝试黑客技术和程序设计。他经常在计算机安全会议上发表演讲，并且在世界各地培训安全人员。目前他在北加利福尼亚州从事漏洞安全的研究工作。

致　谢

感谢 Bill Pollock 以及 No Starch 出版社的每一个人，是他们使本书得以出版，是他们使我得以尽可能多地用创造性的方法控制这一过程。感谢本书的校对、编辑，我的朋友 Seth Benson 和 Aaron Adam。感谢 Jack Matheson 在汇编语言方面给我的帮助。感谢 Dr. Seidel 使我在计算机科学方面一直兴趣浓厚。感谢我的父母为我购买了第一台 Commodore VIC-20 计算机。感谢黑客团体，是他们的新发明和创造力造就了本书讨论的这些技术。

目　　录

第 1 章　简介

hacking 这个概念可能使人们联想到一些固定的形象：恶意的电子破坏行为、间谍行为，以及染发和纹身。大多数人把 hacking 和违法行为联系在一起，并会主观地认为凡是从事 hacking 行为的人都是罪犯。我们承认，虽然有一些使用黑客技术的人违反了法律，但 hacking 本身并非如此。实际上，大多数的黑客是守法的，而不是违法的。hacking 的实质是寻找法律和给定情况下一些道具非计划的或者被忽视的用途，并通过全新的创造性方式应用它们来解决问题——无论是什么问题。

hacking 的实质可以用下面的数学问题来阐明：

使用数字 1、3、4 和 6，且每个只使用一次，和任意 4 个基本数学运算符（加、减、乘和除）得到总和 24。每个数字只能使用一次，您可以规定运算次序：例如公式 3×（4+6）+1=31 是正确的，但却不符合要求，因为该式的总和不是 24。

这个数学问题的规则清晰、简单，但却难倒了很多人。像对该问题的解法一样（参见本书最后一页），黑客的解决办法完全遵循系统的规则，但他们以违反直觉的方式使用这些规则。这种规则的使用方式为黑客赋予了开路利刃，使他们能够以对于那些仅局限于传统思维和方法的人来说不可思议的方式来解决问题。

在计算机发展的初期，黑客就已经能够创造性地解决问题了。在 20 世纪 50 年代后期，MIT 模型铁路俱乐部在这方面做出了贡献，该俱乐部的大多数设备是陈旧的电话设备。俱乐部的会员们使用这样的设备装配出了一个复杂的系统，这个系统允许多个操作员通过拨入合适的区域来控制路轨的不同部分。他们把设备的这种新的创造性用法称为“hacking”，很多人认为这个小组就是黑客的雏形。他们继续为早期的计算机（如 IBM 704 和 TX-0）设计程序，并把这些程序存储在穿孔纸片和纸带上。其他人仅仅满足于编写出解决问题的程序时，早期的黑客们却迷恋于写出能够**很好**地解决问题的程序了。人们认为与已存在的程序相比，能够得到相同的结果但使用较少穿孔卡片的程序比较好，即使这个程序完成的是相同的工作，主要区别在于程序如何获得它的结果——**高雅**。

减少程序所需的穿孔卡片数目显示出了计算机的艺术技巧。一张工艺精致的桌子和一个牛奶包装箱一样都可以托住一个花瓶，但桌子显然看起来要比包装箱好得多。早期的黑客证明技术问题可以有艺术化的解决方法，因此他们将编程从一门纯粹的工程任务转变为一门艺术。

像其他许多艺术形式一样，hacking 经常使人无法理解。那些少数掌握黑客技术的人形成了一个非正规的亚文化群，他们仍然密切关注于学习和掌握他们的艺术。他们认为信息应该是自由的，任何妨碍自由的东西都应该禁止。这些障碍包括掌权者、教育官僚主义和歧视。在以毕业为目标的众多学生中，这个非官方的黑客团体公然挑战传统的获取高分这样的目标，而以获取知识为目的。这使得他们不断地学习和探索，甚至超越了传统的界限。12 岁的 Peter Deutsch 显示出他在 TX-0 方面的知识以及学习的渴望时，MIT 模型铁路俱乐部便接收了他更说明了这

一点。年龄、种族、性别、外表、学位和社会地位不是判断一个人价值的主要标准，这并不是因为希望平等，而是因为希望发展 hacking 这门新兴艺术。

最初的黑客在传统且枯燥的数学和电子学中找到了辉煌和高雅。黑客将编程看作一种艺术表现形式，将计算机看作是艺术创作工具。他们详细研究和理解的愿望并不是打算褪去他们艺术般努力的神秘面纱，只不过是为了使人们对他们更加欣赏。这些知识驱动的价值最终将被称作**黑客道德准则**：欣赏艺术形式的逻辑，促进信息的自由流动，打破传统的界限和束缚，只是为了一个简单的目标，即更好地了解客观世界。早在古希腊时期，毕达哥拉斯就具有类似的道德准则和亚文化群，尽管他并没有计算机。他们发现了数学的美妙之处，同时发现了几何学中的很多核心概念。对知识的渴望以及许多有用的副产品贯穿了整个历史，从毕达哥拉斯到艾达·拉瓦雷斯，到艾伦·图灵，到 MIT 模型铁路俱乐部的黑客们。像 Richard Stallman 和 Steve Wozniak 一样的现代黑客们已经延续了黑客的衣钵，带给我们现代的操作系统、程序设计语言、个人电脑和许多其他我们在天天使用的技术。

那么我们如何辨别哪些是能够给我们带来技术进步奇迹的好黑客，哪些是偷窃我们信用卡号的邪恶黑客呢？人们杜撰出**骇客**这个术语专门形容那些邪恶的黑客，从而将他们与好黑客区分开。新闻工作者曾经被告知**骇客**是一群坏家伙，而黑客则是好的。黑客坚持他们的黑客道德准则，而骇客却对违法的事情和迅速暴富感兴趣。人们认为骇客的才能远远不如精英黑客，这是因为他们仅仅是简单地应用黑客编写的工具和脚本，而没有真正理解它们是如何工作的。**骇客**想要的是肆无忌惮地应用计算机来做那些他们想做的事情——盗用软件、攻击网站以及其他更坏的事情，但他们并不理解他们所做的事。今天几乎已经没有人使用这个术语了。

这个术语没有普及流行的可能原因是它令人困惑的语源——**骇客**这个术语最早用来描述那些破解软件版权和通过反向工程拷贝保护方案的那些人。当下不流行可能仅仅是由于它的两个有分岐的新定义：一群利用电脑图谋不轨的人或技术不高明的**黑客**。大多数新闻工作者都感觉没有必要使用一个读者并不熟悉的术语。相反，大部分人都知道与术语黑客有关的神秘和技巧，因此，对新闻工作者来说，很容易决定使用术语**黑客**。与此相似，人们有时使用术语 script kiddie 来称呼骇客，但它不具有与影子**黑客**相同的名声。有人仍旧坚持认为在黑客和骇客之间有一条明确的分界线，但我认为具有黑客精神的是黑客，不论他或者她违反什么法律。

当前法律限制密码学和密码研究更进一步模糊了黑客和骇客之间的分界线。2001 年，普林斯顿大学的 Edware Felten 教授和他的研究团队发表了一篇讨论各种不同的数字水印方案弱点的论文。这篇文章回应了由安全数字音乐联盟（Secure Digital Music Initiative，SDMI）在 SDMI Public Challenge 上提出的挑战，鼓励公众尝试破解这些水印方案。虽然 Edware Felten 教授和他的研究团队在发表这篇文章之前，受到了来自 SDMI 基金和美国唱片业协会（Recording Industry Association of American，RIAA）的双重威胁。1998 年通过的数字千年版权法案（Digital Millenium Copyright Act，DMCA）使得讨论和提供可能用来绕过行业消费者控制的技术是非法的。人们用同一法律来反对 Dmitry Sklyarov——一位俄罗斯程序员和黑客。他编写软件破解了 Adobe 软件中过分简单的加密，并在美国的一次黑客大会上展示了他的发现。美国国家联邦调查局（FBI）发起突然袭击逮捕了他，导致了长时间的法律抗争。根据法

律规定，行业消费者控制的复杂性无关紧要，如果作为行业消费者控制使用，那么反向工程或者甚至讨论儿童黑话（Pig Latin，一种英语语言游戏）在技术上都可能是非法的。那么现在到底谁是黑客谁是骇客呢？法律看上去似乎要干涉言论自由时，是否一个守法的公民如果说出自己想法就会突然变成一个不守法律的人呢？我认为黑客精神超越了国家的法律，与法律给他们所下的定义正相反。

核物理学和生物化学可以用来杀人，同时也可以给我们带来巨大的科学进步和现代医学。知识本身并没有好坏之分，好坏在于知识的应用。即使我们想，我们也无法抑制技术的成果转化，不能阻挡社会技术的不断进步。同样，黑客精神永远不能被终止，也不能轻易被分类或剖析。黑客将不断推进知识和可接受行为的极限，迫使我们不断地深入探索。

这种推进的一部分最终导致了有益于安全的共同发展，这是通过攻击黑客和防御黑客之间的竞争实现的。正像速度飞快的瞪羚适应猎豹的追逐一样，猎豹会从追逐瞪羚的过程中变得速度更快一些，黑客之间的竞争会给计算机用户提供更好更强大的安全措施，以及更加复杂和完善的攻击技术。入侵检测系统（IDS）的引入和发展就是这种共同发展过程的一个最好例子。防御黑客制作了入侵检测系统并添加到他们的武器库中，而攻击黑客会开发 IDS 规避技术，而这些规避技术又会被更大更好的 IDS 产品所弥补。这种交互作用的最终结果是有益的，因为它会使人更聪明、安全措施更好、软件稳定性更高、解决问题的技巧更有创造性，甚至会带来新经济。

本书的目的是向读者介绍黑客技术的精髓。我们将讨论从过去到现在的各种黑客技术，剖析它们如何作用及其作用原理。本书包含一张可启动光盘（LiveCD），该光盘包含本书使用的所有源代码以及一个预配置好的 Linux 工作平台。探索和创新对黑客艺术来说至关重要，而该 CD 可以使您紧跟本书内容并自己进行实验。唯一的需求是一台 X86 处理器，所有的 Microsoft Windows 机器和较新的 Macintosh 电脑都适用这种处理器，您只需插入光盘并重启系统。该备用 Linux 工作环境不会干扰您现有的操作系统，因此您使用完以后，只需再次重启系统并拿出光盘即可。您会亲身感受并感激 hacking，它可能会激励您改进现有的技术，甚至发明一种新技术。我们希望，本书将激发读者对黑客的好奇天性，促使读者在某些方面对 hacking 艺术有所贡献，不管您选择站在黑客的哪一边。

第 2 章　程序设计

黑客这个术语指那些编写代码和探索代码漏洞的人。尽管这两类黑客的最终目的不同，但这两类人使用的问题求解方法相似。由于对程序设计的理解对发掘漏洞代码的人有帮助，且对漏洞发掘的理解对编写代码的人有帮助，因而，许多黑客既编写代码又发掘代码漏洞。在编写高雅的代码所使用的技巧以及漏洞发掘程序所使用的技巧中都存在有趣的 hack。hacking 实际上恰恰是寻求问题的巧妙的、反直觉的解决方案的一种行为。

在程序漏洞发掘中发现的 hack 通常以意想不到的方式使用计算机规则来绕过安全措施。在编写代码中发现的 hack 与此类似，这是因为它们也是以新颖的、创造性的方法使用计算机规则，但编写程序的最终目标是追求高效或者使用更少的源代码，而不一定出于安全方面的考虑。实际上，为完成某项给定任务可以编写无穷多的程序，但是这些解决方案中的大多数过分庞大、复杂且随意。只有少数几个解决方案小巧、效率高且简洁。程序的这种特有的品质称为**高雅**，巧妙的、富有创造性的、并且有助于产生这样高效率的解决方案称作 hack。两个方面的黑客编程往往既欣赏高雅代码的美妙又欣赏巧妙的 hack 的独创性。

在商业世界中，人们更注重于写出实用的代码而不是获得巧妙的 hack 和简洁的代码。因为计算能力和内存惊人地以指数律增长，所以在使用拥有千兆赫兹处理周期和千兆字节内存的现代计算机时，额外花费 5 小时创建一段速度稍快并且内存利用效率更高的代码没有任何商业价值。虽然时间和内存的优化除了引起那些最高端的用户注意之外，并没有引起其他任何人的注意，但是这样的机器很畅销。当底线是金钱时，为优化程序而花费很多时间在巧妙的 hack 上是没有意义的。

对程序设计高雅性的欣赏留给了黑客：计算机爱好者的最终目的不是获利，而仅仅是尽他们所能扩展老式 Commodore 64 的每一点功能，漏洞发掘的作者需要编写很少量但令人惊叹的代码来溜过狭窄的安全缝隙，而其他人也欣赏这种追求和寻找最佳可能解决方案的精神。这些就是对编程感到兴奋，并且真正欣赏高雅代码的魅力和巧妙 hack 的创新性的人。因为理解程序设计是理解程序如何被利用的先决条件，所以起点自然是程序设计。

2.1　什么是程序设计

程序设计是一个非常自然和直观的概念。程序仅仅是用某种具体的语言写出的一系列语句。程序随处可见，即使是在对技术充满恐惧的社会每天也会用到程序。驾驶路线说明、烹饪菜谱、足球比赛以及 DNA 都是形形色色的程序。一个典型的驾驶路线说明程序看上去可能是这样描述的：

从标有 Main Street 标志的大街出发向东，一直向前走，直到看到右边有一座教堂。如果由于有建筑物而堵塞了道路，则从那儿右转到第 15 大街，向左转到 pine 大街，然后向右转到

第 16 大街。否则，您可以仅仅向右转到第 16 大街。继续沿着第 16 大街走，向左转到 Destination 路。沿着 Destination 路走 5 英里，右边有一座房子，地址是 Destination 路 743 号。

任何懂英语的人都能明白并按照驾驶路线的说明到达目的地，因为驾驶路线说明是用英语描述的。的确，这些描述并不是很精彩，但每一步都非常清楚且很容易理解，至少对一个懂英语的人而言是这样的。

但是计算机生来并不懂英语，它只能理解机器语言。为了指导计算机做某事，指令必须用机器语言写出。然而，通晓**机器语言**的人很少且机器语言难以使用——机器语言由原始的比特和字节组成，而且随着计算机体系结构的不同而变化。所以，为了用机器语言给 Intel x86 处理器编写一个程序，人们不得不计算好与每一条指令相对应的值，了解各条指令之间如何交互，以及许多其他底层的细节。像这样的程序设计是一项辛苦的、麻烦的工作，当然也不是凭直觉就可以完成的。

克服编写机器语言程序困难唯一需要的是一个翻译程序。**汇编程序**是一种形式的机器语言翻译程序——它是一个把汇编语言程序翻译成计算机可以识别的代码的程序。汇编语言没有机器语言那么神秘，因为汇编语言为不同的指令和变量命名，而不仅仅是使用一些数字。然而，汇编语言还远远不是直观的语言。汇编语言的指令名非常深奥，并且汇编语言仍然与计算机的体系结构密切相关。这意味着，正像 Intel x86 处理器的机器语言与 Sparc 处理器的机器语言不同一样，x86 的汇编语言也不同于 Sparc 的汇编语言。利用汇编语言为某种处理器体系结构编写的任何程序在另一种处理器的体系结构上都不能运行。如果某个程序是用 x86 汇编语言编写的，如果想让它在 Sparc 的体系结构上运行则必须重写。此外，为了用汇编语言写出高效率的程序，人们还必须懂得许多关于那种处理器的底层的细节。

这些问题也可以通过另一种称为**编译程序**的翻译程序来进一步解决。编译程序可以把高级语言翻译成机器语言。高级语言比汇编语言更直观，并且能够翻译成适合各种不同体系结构处理器的不同类型的机器语言。这意味着如果一个程序是用高级语言编写的，则程序只需编写一次，相同的一段程序代码通过编译程序可以被编译为适用各种具体体系结构处理器的机器语言。C、C++和 FORTRAN 都是高级语言的例子。用高级语言编写的程序比汇编语言或机器语言可读性更好，更像英语。但是高级语言仍然必须遵守非常严格的指令规则，否则编译程序将无法理解。

2.2　伪代码

程序员还有另外一种形式的程序设计语言——伪代码。**伪代码**是用类似高级语言的一般结构形式组织起来的简单的英语。它不能被编译程序、汇编程序或者任何计算机所理解，但对程序员来说伪代码是一种非常有用的组织指令的方法。伪代码没有明确的定义。实际上，许多人写的伪代码都有一点差异。伪代码是自然语言（例如英语）和类似于 C 一样的高级程序设计语言之间有几分模糊不清并缺少的一环。伪代码是常见的通用程序设计概念的出色入门工具。

2.3 控制结构

如果没有控制结构，那么一段程序就仅仅是一系列按排列顺序执行的指令。对于简单程序来说，顺序执行方式很好，但大多数程序，像驾驶路线说明这样的例子，并不像这样简单。驾驶路线说明包括类似这样的语句：**在 Main Street 大街上一直向前走，直到看到右边有一座教堂，如果有建筑物堵塞了道路**……我们将这些语句称为**控制结构**，它们将程序执行的流向由简单的顺序执行改变为更复杂、更有用的流向。

2.3.1 If-Then-Else

在驾驶路线说明这个例子中，Main Street 可以受结构控制。如果结果为真，就需要一组专门指令来应用于这种情况。否则，应当继续执行原来的指令集。可以用一种最自然的控制结构来解决程序中这种类型的实例：即 If-Then-Else *结构*。通常，它看起来像这样：

```
If (condition) then
{
    Set of instructions to execute if the condition is met;
}
Else
{
    Set of instruction to execute if the condition is not met;
}
```

在本书中，将会使用一种类 C 伪代码，因此每条指令会以分号（;）结束，并且会用大括号及缩进来对指令集进行分组。前面驾驶路线说明的 If-Then-Else 伪代码结构看起来像这样：

```
Drive down Main Street;
If (street is blocked)
{
    Turn right on 15th Street;
    Turn left on Pine Street;
    Turn right on 16th Street;
}
Else
{
    Turn right on 16th Street;
}
```

每条指令占一行，并且各个条件指令集以大括号和缩进分组以便于阅读。在 C 以及许多其他程序设计语言中，Then 关键字是隐含值常被省去，所以在前面的伪代码中也省略了 Then。

当然，其他一些程序设计语言在其语法中要求有 then 关键字，例如 BASIC、FORTRAN，甚至 Pascal。在程序设计语言中，这些类型的语法差异只是存在于表面上，它们的底层结构仍然一样。一旦程序员理解了这些语言想要传达的思想，学习各种不同的语法变体是毫不费力的。由于在随后的章节中会用到 C 语言，因此本书中使用的伪代码遵循类 C 语法规则，但是请记住伪代码能够以多种形式呈现。

另一个类 C 语法的常见规则是：被大括号包含的一组指令只有一条指令时，大括号是可选的。为了便于阅读，缩进指令仍然是一个不错的主意，但语法上并没有这种要求。可以遵循该规则将前面的驾驶路线说明示例重写，会产生如下的等价伪代码段：

```
Drive down Main Street;
If (street is blocked)
{
    Turn right on 15th Street;
    Turn left on Pine Street;
    Turn right on 16th Street;
}
Else
    Turn right on 16th Street;
```

这个有关指令集的规则对于本书中提到的所有控制结构都是有效的，并且规则本身也可以用伪代码描述。

```
If (there is only one instruction in a set of instructions)
    The use of curly braces to group the instructions is optional;
Else
{
    The use of curly braces is necessary;
    Since there must be a logical way to group these instructions;
}
```

甚至对语法本身的描述也可以认为是一个简单的程序。此外，还存在着 If-Then-Else 结构的变体，例如 Select/Case 语句，但其根本逻辑是一样的：如果这件事发生了，则做这些事，否则做其他的事（甚至可以是更多的 If-Then 语句）。

2.3.2　While/Until 循环

另一个基本的程序设计概念是 While 控制结构，它是一种循环类型。程序员经常想要多次执行同一组指令。程序可以通过循环来完成这项任务，但是它要求一组条件告诉它什么时候停止循环，以免它无限继续下去。条件为真时，**While 循环**会指明在一个循环中执行随后的指令集。一个为饥饿的老鼠编写的简单程序如下所示：

```
While (you are hungry)
{
    Find some food;
    Eat the food;
}
```

老鼠仍然饥饿时，紧随 while 语句的两行指令会被重复执行。老鼠每次找到的食物数量可能从一小块面包屑到一整块面包不等。与此类似，While 语句中指令集的执行次数的变化取决于老鼠找到的食物的多少。

While 循环的另一种变体是 Until 循环，Until 循还是 Perl 程序设计语言中一种有效的语法（C 语言不使用这种语法）。**Until 循环**其实是条件语句颠倒的 While 循环。使用 Until 循环的与前面相同的老鼠程序如下所示：

```
Until (you are not hungry)
{
    Find some food;
    Eat the food;
}
```

从逻辑上来说，任何 Until 循环语句都可以变为 While 循环。前面提到的驾驶路线说明包含一条语句：**在 Main Street 一直向前，直到看到右边有一座教堂**。只需将条件颠倒，就可以将这条语句改变为一个标准的 While 循环。

```
While (there is not a church on the right)
    Drive down Main Street;
```

2.3.3 For 循环

另一个循环控制结构是 **For 循环**。该循环一般用于程序员想要执行确定次数的重复操作时。驾驶路线说明“**沿着 Destination 路走 5 英里**”可以转换成如下所示的 For 循环：

```
For (5 iterations)
    Drive straight for 1 mile;
```

事实上，For 循环只不过是一个有计数的 While 循环。同样的语句可以像这样来写：

```
Set the counter to 0;
While (the counter is less than 5)
{
    Drive straight for 1 mile;
    Add 1 to the counter;
}
```

类 C 伪代码语法的 For 循环计数更加明显：

```
For (i=0; i<5; i++)
  Drive straight for 1 mile;
```

在这个例子中，计数器名为 i，For 语句分解为由分号分隔的三部分。第一部分声明计数器并设置它的初始值，在本例中为 0。第二部分像是一个使用计数器的 While 语句：计数器满足这个条件，就继续循环。第三部分即最后一部分说明在每次循环过程中应当对计数器采取什么动作。在这个例子中，i++是**向名为 i 的计数器加 1** 的简写形式。

利用所有控制结构，可以将 2.1 节中的驾驶路线说明转换成如下所示的类 C 伪代码：

```
Begin going East on Main Street;
While (there is not a church on the right)
  Drive down Main Street;
If (street is blocked)
{
  Turn right on 15th Street;
  Turn left on Pine Street;
  Turn right on 16th Street;
}
Else
  Turn right on 16th Street;
Turn left on Destination Road;
```