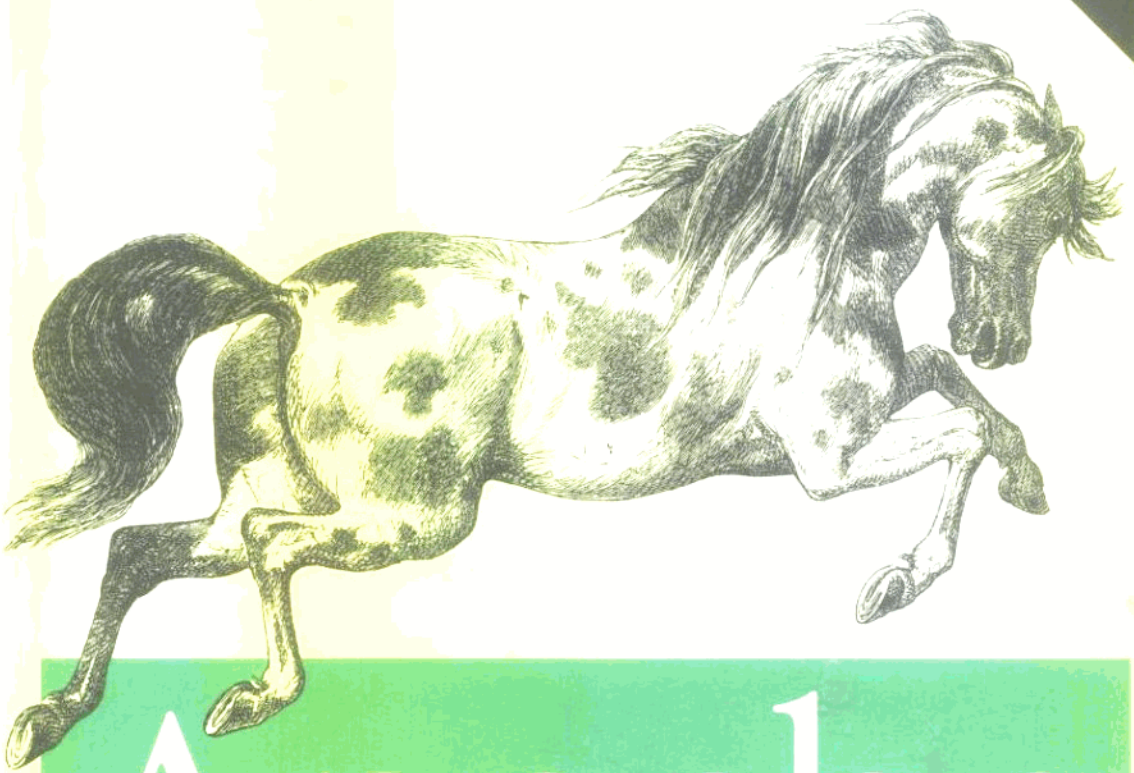
Apache权威指南

（影印版）

第二版
含光盘



Apache

The Definitive Guide

O'REILLY®
中国电力出版社

Ben Laurie & Peter Laurie

Desktop Reference

Apache 1.3.4

Apache

Quick Reference

O'REILLY®

Andrew Ford

图书在版编目（CIP）数据

实用C语言编程/（美）奥莱恩著；郭大海译. -北京：中国电力出版社，2000.5
书名原文：Practical C Programming
ISBN 7-5083-0308-3

Ⅰ.实… Ⅱ.①奥… ②郭… Ⅲ.C语言程序设计 Ⅳ.TP312
中国版本图书馆CIP数据核字（2000）第08994号

北京市版权局著作权合同登记
图字：01-1999-3746号

书　　名/ 实用C语言编程
书　　号/ ISBN 7-5083-0308-3
责任编辑/ 刘君，关敏，蒙虎
封面设计/ Ellie Volckhausen，Hanna Dyer，张健
出版发行/ 中国电力出版社
地　　址/ 北京三里河路6号（邮政编码100044）
经　　销/ 全国新华书店
印　　刷/ 北京市地矿印刷厂
开　　本/ 787毫米×1092毫米　16开本　31印张　310千字
版　　次/ 2000年5月第一版　2000年5月第一次印刷
印　　数/ 0001-5000册
定　　价/ 49.00元（册）

前言

本书讲述如何使用C语言进行真正的编程。C是目前软件开发者们最主要的编程语言。这也是它受到广泛传播并且成为标准的原因。现在也有新的编程语言出现，如C++,但这些语言仍然在演化中。C仍然是进行健壮的、可移植编程的首选语言。

本书侧重介绍在实际编程过程中需要知道的技巧问题。它不仅告诉你有关C语言的机制，也告诉你用C进行编程的整个过程（包括程序的含义、设计、编码、方法、调试、发布、文档、维护和版本更新等）。

本书也介绍了用C编程的风格和艺术。要写出一个好的程序，需要做许多工作，而不仅仅是敲一些代码。这是一门有关写作和编程的技巧、并将其合二为一的艺术，从而使你写出的程序成为一件杰作。这样创作出来的作品才是真正的艺术。一个非常好的程序不仅仅要功能正确，而且应该简单易读。在程序中允许加进一些评论性的描述性文字。当清晰的评注被加进程序中时，这样的程序就会得到其他人的高度评价。

程序应该尽可能简单。程序员应该避免玩一些聪明的技巧。在本书中我们强调简单、实用的原则。例如，在C中有15条关于操作符的规则，但它们可以被简化为下面两个：

1. 先乘除，后加减。

2. 括号优先于其他一切运算。

考虑两个程序：一个是由聪明的程序员使用了所有技巧编写的程序，程序没有包含任何注解，但是可以运行；另外一个程序有很好的注解和结构，但是它不能运行。两个程序哪一个更有用呢？从长远的角度看，应该是那个不能运行的程序，因为它存在的问题可以被解决掉。尽管那个聪明的程序员的程序现在可以运行，但迟早该程序会被修改的。那么到那时，最坏的事情就会出现 —— 你不得不去修改一个充满技巧、但没有任何注解的程序。

这本书是针对那些没有编程经验，或者已经对C语言有所了解但是还想进一步提高他们的编程风格和程序可靠性的人所作的。在使用这本书之前，你应该知道如何使用计算机并且知道如何使用一些基本的工具，诸如文本编辑器和文件系统。

对于那些想在UNIX操作系统上用generic cc编译器或者自由软件基金会的gcc编译器编写和运行程序的读者，我们给出了一些特别的指导。对于MS-DOS/Windows的用户，我们也给出了包括Borland C++, Turbo C++和Microsoft Visual C++的使用说明(这些编译器可以对C和C++的代码进行编译)。在本书中也给出了一些使用编程工具make进行自动程序编译的例子。

这本书是如何组织的

在你学会跑步之前，你必须先学会走路。

在第一部分 —— 基础中，你将会学习如何走路。本章只讲述如何编写一个非常简单的程序。读者从编程技巧和编程风格入手。下一步，你将学习如何使用变量和非常简单的判断和控制语句。在第七章程序设计过程中，我们才将编写真正程序的完整过程展现给读者。

在第二部分 —— 简单程序设计中，我们介绍编程时所有需要的简单语句和运算符。读者还将学习到如何把这些语句组织成为简单函数的方法。

在第三部分 —— 高级编程概念中，我们将介绍一些基本的说明和语句来构造高

目录

1

Getting Started

When you connect to the URL of someone's home page—say the notional *http://www.butterthlies.com/* we shall meet later on—you send a message across the Internet to the machine at that address. That machine, you hope, is up and running, its Internet connection is working, and it is ready to receive and act on your message.

URL stands for Universal Resource Locator. A URL such as *http://www.butter-thlies.com/* comes in three parts:

```
<method>://<host>/<absolute path URL (apURL)>
```

So, in our example, *<method>* is `http`, meaning that the browser should use HTTP (Hypertext Transfer Protocol); *<host>* is `www.butterthlies.com`; and *<apURL>* is "`/`", meaning the top directory of the host. Using HTTP/1.1, your browser might send the following request:

```
GET / HTTP/1.1
Host: www.butterthlies.com
```

The request arrives at port 80 (the default HTTP port) on the host *www.butterthlies.com*. The message is again in three parts: a method (an HTTP method, not a URL method), that in this case is `GET`, but could equally be `PUT`, `POST`, `DELETE`, or `CONNECT`; the Uniform Resource Identifier (URI) "`/`"; and the version of the protocol we are using. It is then up to the web server running on that host to make something of this message.

It is worth saying here—and we will say it again—that the whole business of a web server is to translate a URL either into a filename, and then send that file back over the Internet, or into a program name, and then run that program and send its output back. That is the meat of what it does: all the rest is trimming.

The host machine may be a whole cluster of hypercomputers costing an oil sheik's ransom, or a humble PC. In either case, it had better be running a web server, a program that listens to the network and accepts and acts on this sort of message.

What do we want a web server to do? It should:

- Run fast, so it can cope with a lot of inquiries using a minimum of hardware.
- Be multitasking, so it can deal with more than one inquiry at once.
- Be multitasking, so that the person running it can maintain the data it hands out without having to shut the service down. Multitasking is hard to arrange within a program: the only way to do it properly is to run the server on a multitasking operating system. In Apache's case, this is some flavor of Unix (or Unix-like system), Win32, or OS/2.
- Authenticate inquirers: some may be entitled to more services than others. When we come to virtual cash, this feature (see Chapter 13, *Security*) becomes essential.
- Respond to errors in the messages it gets with answers that make sense in the context of what is going on. For instance, if a client requests a page that the server cannot find, the server should respond with a "404" error, which is defined by the HTTP specification to mean "page does not exist."
- Negotiate a style and language of response with the inquirer. For instance, it should—if the people running the server can rise to the challenge—be able to respond in the language of the inquirer's choice. This ability, of course, can open up your site to a lot more action. And there are parts of the world where a response in the wrong language can be a bad thing. If you were operating in Canada, where the English/French divide arouses bitter feelings, or in Belgium, where the French/Flemish split is as bad, this feature could make or break your business.
- Offer different formats. On a more technical level, a user might want JPEG image files rather than GIF, or TIFF rather than either of the former. He or she might want text in vdi format rather than PostScript.
- Run as a proxy server. A proxy server accepts requests for clients, forwards them to the real servers, and then sends the real servers' responses back to the clients. There are two reasons why you might want a proxy server:
  - The proxy might be running on the far side of a firewall (see Chapter 13), giving its users access to the Internet.
  - The proxy might cache popular pages to save reaccessing them.