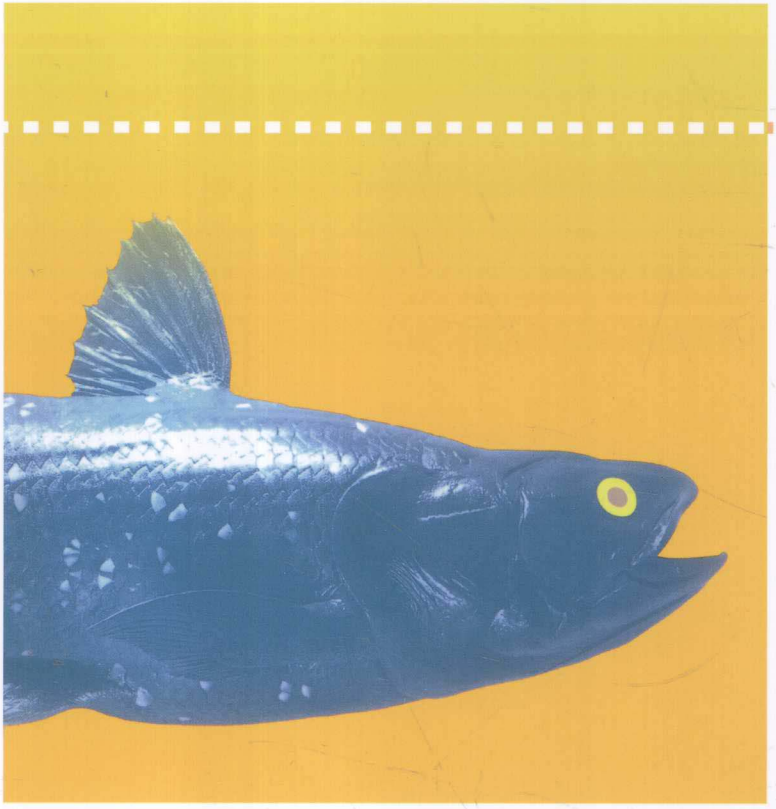
PEARSON

C 语言经典著作　聆听大师教诲　掌握编程精髓



C专家编程

（英文版）

Expert C Programming

Deep C Secrets

〔美〕Peter van der Linden　著

人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

C专家编程

（英文版）

Expert C Programming

Deep C Secrets

〔美〕Peter van der Linden 著

人民邮电出版社
北京

序

最近，我逛了一家书店，当我看到大量枯燥乏味的 C 和 C++书籍时，心情格外沮丧。我发现极少有作者想向读者传达这样一个信念：任何人都可以享受编程。在冗长而乏味的阅读过程中，所有的奇妙和乐趣都烟消云散了。如果你硬着头皮把它啃完，或许会有长进。但编程本来不该是这个样子的呀！

编程应该是一项精妙绝伦、充满生机、富有挑战的活动，而讲述编程的书籍也应时时迸射出激情的火花。本书也是一本教学性质的书籍，但它希望重新把快乐融入编程之中。如果本书不合你的口味，请把它放回到书架上，但务必放到更显眼的位置上，这里先谢过了。

好，听了这个开场白，你不免有所疑问：关于 C 语言编程的书可以说是不胜枚举，那么这本书又有什么独到之处呢？

《C 专家编程》应该是每位程序员的第二本学习 C 语言的书。这里所提到的绝大多数教程、提示和技巧都是无法在其他书上找到的，即使有的话，它们通常也是作为心得体会手工记录在手册的书页空白处或旧打印纸的背面。作者以及 Sun 公司编译器和操作系统小组的同事们在多年 C 语言编程实践中，积累了大量的知识和经验。书中讲述了许多有趣的 C 语言故事和轶闻，诸如连接到因特网上的自动售货机、太空软件中存在的问题，以及一个 C 语言的缺陷怎样使整个 AT&T 长途电话网络瘫痪等。本书的最后一章是 C++语言的轻松教程，帮助你精通这门日益流行的从 C 语言演化而来的语言。

本书讲述的是应用于 PC 和 UNIX 系统上的 ANSI 标准 C 语言。对 C 语言中与 UNIX 平台复杂的硬件结构（如虚拟内存等）相关的特性作了详细描述。对于 PC 的内存模型和 Intel 8086 系列对 C 语言产生影响的部分也作了全面介绍。C 语言基础相当扎实的人很快就会发现书中充满了很多程序员可能需要多年实践才能领会的技巧、提示和捷径。它覆盖了许多令 C 程序员困惑的主题：

- typedef struct bar{ int bar; }bar 的真正意思是什么？
- 我怎样把一些大小不同的多维数组传递到同一个函数中？
- 为什么 extern char *p;同另一个文件的 char p[100];不能够匹配？

- 什么是总线错误（bus error）？什么是段违规（segmentation violation)?
- char *foo[]和 char(*foo)[]有何不同？

如果你对这些问题不是很有把握，很想知道 C 语言专家是如何处理它们的，那么请继续阅读！即使你对这些问题已经了如指掌，对 C 语言的其他细节也是耳熟能详，那么也请阅读本书，继续充实你的知识。如果觉得不好意思，就告诉书店职员“我是为朋友买书。”

Peter Van Der Linden 于加州硅谷

前言

C 代码。C 代码运行。运行码运行...请！

——*Barbara Ling*

所有的 C 程序都做同一件事，观察一个字符，然后啥也不干。

——*Peter Weinberger*

你是否注意到市面上存有大量的 C 语言编程书籍，它们的书名具有一定的启示性，如：*C Traps and Pitfalls*（本书中文版《C 陷阱与缺陷》已由人民邮电出版社出版），*The C Puzzle Book, Obfuscated C and Other Mysteries*，而其他的编程语言好像没有这类书。这里有一个很充分的理由！

C 语言编程是一项技艺，需要多年历练才能达到较为完善的境界。一个头脑敏捷的人很快就能学会 C 语言中基础的东西。但要品味出 C 语言的细微之处，并通过大量编写各种不同程序成为 C 语言专家，则耗时甚巨。打个比方说，这是在巴黎点一杯咖啡与在地铁里告诉土生土长的巴黎人该在哪里下车之间的差别。本书是一本关于 ANSI C 编程语言的高级读本。它适用于已经编写过 C 程序的人，以及那些想迅速获取一些专家观点和技巧的人。

编程专家在多年的实践中建立了自己的技术工具箱，里面是形形色色的习惯用法、代码片段和灵活掌握的技巧。他们站在其他更有经验的同事的肩膀上，或是直接领悟他们的代码，或是在维护其他人的代码时聆听他们的教诲，随着时间的推移，逐步形成了这些东西。另外一种成为 C 编程高手的途径是自省，在认识错误的过程中进步。几乎每个 C 语言编程新手都曾犯过下面这样的书写错误：

```
if(i = 3)
```

正确的应该是：

```
if(i == 3)
```

一旦有过这样的经历，这种痛苦的错误（需要进行比较时误用了赋值符号）一般不会再犯。有些程序员甚至养成了一种习惯，在比较式中先写常数，如：if(3 == i)。这样，如果不小心误

用了赋值符号，编译器就会给出“attempted assighnment to literal（试图向常数赋值）”的错误信息。虽然当你比较两个变量时，这种技巧起不了作用。但是，积少成多，如果你一直留心这些小技巧，迟早会对你有所帮助的。

价值 2000 万美元的 Bug

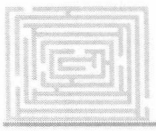
1993 年春天，在 SunSoft 的操作系统开发小组里，我们接到了一个“一级优先”的 Bug 报告，是一个关于异步 I/O 库的问题。如果这个 Bug 不解决，将会使一桩价值 2000 万美元的硬件产品生意告吹，因为对方需要使用这个库的功能。所以，我们顶着重压寻找这个 Bug。经过几次紧张的调试，问题被圈定在下面这条语句上：

```
x == 2;
```

这是个打字错误，它的原意是一条赋值语句。程序员的手指放在“=”键上，不小心多按了一下。这条语句成了将 x 与 2 进行比较，比较结果是 true 或者 false，然后丢弃这个比较结果。

C 语言的表达能力也实在是强，编译器对于“求一个表达式的值，但不使用该值”这样的语句竟然也能接受，并且不发出任何警告，只是简单地把返回结果丢弃。我们不知道是应该为及时找到这个问题的好运气而庆幸，还是应该为这样一个常见的打字错误可能付出高昂的代价而痛心疾首。有些版本的长整数程序已经能够检测到这类问题，但人们很容易忽视这些有用的工具。

本书收集了其他许多有益的故事。它记录了许多经验丰富的程序员的智慧，避免读者再走弯路。当你来到一个看上去很熟的地方，却发现许多角落依然陌生，本书就像是一个细心的向导，帮助你探索这些角落。本书对一些主要话题如声明、数组/指针等作了深入的讨论，同时提供了许多提示和记忆方法。本书从头到尾都采用了 ANSI C 的术语，在必要时我会用日常用语来诠释。



编程挑战

小 启 发

样例框

我们设置了“编程挑战”这个小栏目，像这样以框的形式出现。
框中会列出一些对你所编写的程序的建议。

另外，我们还设置了“小启发”这个栏目，也是以框的形式出现的。

“小启发”里出现的是在实际工作中所产生一些想法、经验和指导方针。你可以在编程中应用它们。当然，如果你觉得你已经有了更好的指导原则，也完全可以不理会它们。

约定

我们所采用的一个约定是用蔬菜和水果的名字来代表变量的名字（当然只适用于小型程序片段，现实中的程序不可如此）：

```
char pear[40];
double peach;
int mango = 13;
long melon = 2001;
```

这样就很容易区分哪些是关键字，哪些是程序员所提供的变量名。有些人或许会说，你不能拿苹果和桔子作比较。但为什么不行呢？它们都是在树上生长、拳头大小、圆圆的可食之物。一旦你习惯了这种用法，你就会发现它很有用。另外还有一个约定，有时我们会重复某个要点，以示强调。

和精美食谱一样，《C 专家编程》准备了许多可口的东西，以实例的样式奉献给读者。每一章都分成几个彼此相关而又独立的小节。无论是从头到尾认真阅读，还是随意翻开一章选一个单独的主题细细品味，都是相当容易的。许多技术细节都蕴藏于 C 语言在实际编程中的一些真实故事里。幽默对于学习新东西是相当重要的，所以我在每一章都以一个“轻松一下”的小栏目结尾。这个栏目包含了一个有趣的 C 语言故事，或是一段软件轶闻，让读者在学习的过程中轻松一下。

读者可以把本书当作 C 语言编程的思路集锦，或是 C 语言提示和习惯用法的集合，也可以从经验丰富的编译器作者那里汲取营养，更轻松地学习 ANSI C。总之，它把所有的信息、提示和指导方针都放在一个地方，让你慢慢品味。所以，请赶紧翻开书，拿出笔，舒舒服服在坐在电脑前，开始快乐之旅吧！

轻松一下——优化文件系统

偶尔，在 C 和 UNIX 中，有些方面是令人感觉相当轻松的。只要出发点合理，什么样的奇思妙想都不为过。IBM/Motorola/Apple PowerPC 架构具有一种 E.I.E.I.O 指令[1]，代表“Enforce In-Order Execution of I/O”（在 I/O 中实行按顺序执行的方针）。与这种思想相类似，在 UNIX 中也有一条称作 tunefs 的命令，高级系统管理员用它修改文件系统的动态参数，并优化磁盘中文

[1] 可能是由一个名叫 McDonald 的老农设计的。

件块的布局。

和其他的 Berkeley[2]命令一样，在早期的 tunefs 在线手册上，也是以一个标题为“Bugs”的小节来结尾。内容如下：

Bugs:

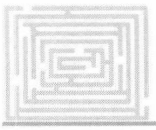
这个程序本来应该在安装好的（mounted）和活动的文件系统上运行，但事实上并非如此。因为超级块（superblock）并不是保持在高速缓冲区中，所以该程序只有当它运行在未安装好的（dismounted）文件系统中时才有效。如果运行于根文件系统，系统必须重新启动。

你可以优化一个文件系统，但不能优化一条鱼。

更有甚者，在文字处理器的源文件中有一条关于它的注释，警告任何人不得忽视上面这段话！内容如下：

如果忽视这段话，你就等着烦吧。一个 UNIX 里的怪物会不断地纠缠你，直到你受不了为止。

当 SUN 和其他一些公司转到 SVr4 UNIX 平台时，我们就看不到这条警训了。在 SVr4 的手册中没有了“Bugs”这一节，而是改名为“注意”（会不会误导大家？）。“优化一条鱼”这样的妙语也不见了。作出这个修改的人现在一定在受 UNIX 里面怪物的纠缠，自作自受！



编程挑战

计算机日期

关于 time_t，什么时候它会到达尽头，重新回到开始呢？

写一个程序，找出答案。

1. 查看一下 time_t 的定义，它位于文件/user/include/time.h 中。

2. 编写代码，在一个类型为 time_t 的变量中存放 time_t 的最大值，然后把它传递给 ctime()函数，转换成 ASCII 字符串并打印出来。注意 ctime()函数同 C 语言并没有任何关系，它只表示“转换时间”。

如果程序设计者去掉了程序的注释，那么多少年以后，他不得不担心该程序会在 UNIX 平台上溢出。请修改程序，找出答案。

1. 调用 time()获得当前的时间。

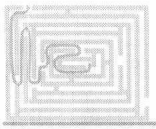
2. 调用 difftime()获得当前时间和 time_t 所能表示的最大时间值之间的差值（以秒计算）。

3. 把这个值格式化为年、月、周、日、小时、分钟的形式，并打印出来。

它是不是比一般人的寿命还要长？

---

[2] 加州大学伯克利分校，UNIX 系统的许多版本都是在那里设计的。 ——译者注



解决方案

计算机日期

这个练习的结果在不同的 PC 和 UNIX 系统上有所差异，而且它依赖于 time_t 的存储形式。在 Sun 的系统中，time_t 是 long 的 typedef 形式。我们所尝试的第一个解决方案如下：

```
#include <stdio.h>
#include <time.h>

int main() {
  time_t biggest= 0x7FFFFFFF;

  printf("biggest = %s \n", ctime(&biggest));
  return 0;
}
```

这是一个输出结果：

```
  biggest = Mon Jan 18 19:14:07 2038
```

显然，这不是正确的结果。ctime()函数把参数转换为当地时间，它跟世界统一时间 UTC（格林尼治时间）并不一致，取决于你所在的时区。本书写作地是加利福尼亚，比伦敦晚 8 个小时，而且现在的年份跟最大时间值的年份相差甚远。

事实上，我们应该采用 gmtime()函数来取得最大的 UTC 时间值。这个函数并不返回一个可打印的字符串，所以不得不用 asctime()函数来获取一个这样的字符串。权衡各方面情况后，修订过的程序如下：

```
#include<stdio.h>
#include<time.h>

int main() {
  time_t  biggest = 0x7FFFFFFF;
  printf("biggest = %s \n", asctime(gmtime(&biggest)));
  return 0;
}
```

它给出了如下的结果：

```
biggest = Tue Jan 19 03:14:07 2038
```

看！这样就挤出了 8 个小时。

但是，我们并未大功告成。如果你采用的是新西兰的时区，你就会又多出 13 个小时，前提是它在 2038 年仍然采用夏令时。他们在 1 月份时采用的是夏令时，因为新西兰位于南半球。但是，由于新西兰的最东端位于日界线的东面，在那里它应该比格林尼治时间晚 10 小时而不是早 14 小时。这样，新西兰由于其独特的地理位置，不幸成为该程序的第一个 Bug 的受害者。

即使像这样简单的问题也可能在软件中埋下令人吃惊的隐患。如果有人觉得对日期进行编程是小菜一碟，一次动手便可轻松搞定，那么他肯定没有深入研究问题，程序的质量也可想而知。

Contents

C Through the Mists of Time 1

C is quirky, flawed, and an enormous success.

—Dennis Ritchie

the prehistory of C...the golden rule for compiler-writers...
early experiences with C...the standard I/O library and C preprocessor...
K&R C...the present day: ANSI C...it's nice, but is it standard?...
the structure of the ANSI C standard...reading the ANSI C standard for
fun, pleasure, and profit...how quiet is a "quiet change"?...some light
relief—the implementation-defined effects of pragmas

The Prehistory of C

The story of C begins, paradoxically, with a failure. In 1969 the great Multics project—a joint venture between General Electric, MIT, and Bell Laboratories to build an operating system—was clearly in trouble. It was not only failing to deliver the promised fast and convenient on-line system, it was failing to deliver anything usable at all. Though the development team eventually got Multics creaking into action, they had fallen into the same tarpit that caught IBM with OS/360. They were trying to create an operating system that was much too big and to do it on hardware that was much too small. Multics is a treasure house of solved engineering problems, but it also paved the way for C to show that small is beautiful.

As the disenchanted Bell Labs staff withdrew from the Multics project, they looked around for other tasks. One researcher, Ken Thompson, was keen to work on another operating system, and made several proposals (all declined) to Bell management. While waiting on official approval, Thompson and co-worker Dennis Ritchie amused themselves porting Thompson's "Space Travel" software to a little-used PDP-7. Space Travel simulated the major bodies of the solar system, and displayed them on a graphics screen along with a space craft that could be piloted and landed on the various planets. At the same time, Thompson worked intensively on providing the PDP-7 with the rudiments of a new operating system, much simpler and lighter-weight than Multics. Everything was