

引言

几年前在一次偶然翻阅 Donald Knuth 所著《TEX: The Program》一书时，序言中的一段话深深触动了我：

我确信 TEX 的最后一个错误已经在 1985 年 11 月 27 日被发现并排除掉了。但是如果出于目前尚不知道的原因，TEX 仍然潜伏有错误，我非常愿意付给第一个发现者\$20.48 元。（这一金额已是以前的两倍。我打算在本年内再增加一倍。你看我是多么自信！）

我对 Knuth 是否曾经付给某人\$20.48 甚至\$40.96 元不感兴趣，这并不重要。重要的是他对他的程序所具有的那种自信。那么据你所知，究竟有多少程序员会严肃地声称他们的程序完全没有错误？又有多少敢把这一声称印刷在书上，并准备为错误的发现者付钱呢？

如果程序员确信测试组已经发现了所有的错误，那么他也许敢作这种声明。但这本身就是一个问题。每当代码被打包装送给程序经销商之前，人们在胸前划着十字带着最好的愿望说：“希望测试已经发现了所有的错误”。这一情景我已见过多次了。

由于现代的程序员已经放弃了对代码进行彻底测试的职责，他们没法知道代码中是否有错。管理人员也不会公布测试情况，只是说：“别操那个心，测试人员会为你作好测试的”。更为微妙的是，管理人员希望程序员自己进行代码的测试。同时，他们希望测试员作得更彻底些，因为毕竟这是他们的本职工作。

正如你在本书中将会看到的那样，编写无错代码的技术或许有几百种，程序员能用，但测试人员却无法使用，因为这些技术和代码的编写直接相关。

两个关键的问题

本书介绍的所有诀窍是当发现错误时，不断地就以下两个问题追问自己的结果：

- 怎样才能自动地查出这个错误？
- 怎样才能避免这个错误？

第一个问题可能使读者认为本书是有关测试的书，其实不是。当编辑程序发现语法错误时，它是在做测试吗？不，不是。编辑程序只是在自动地检查代码中的错误。语法错误只是程序员可以使用的自动查错方法查出的一种最基本的错误类型。本书将详尽介绍自动向程序员提示错误的方法。

编写无错代码的最好方法是把防上错误放在第一位。关于这个问题，同样也有许多的技巧。某些技巧与常用的编码惯例有关，但它们不是象“每个人都违背原则”或“没有人违背该原则”这样泛泛地考虑问题，而是对相应的细节进行详细的讨论。要记住，在任何时候跟在大多数人的后面常常是所能选择的最坏一条路。因此在成为别人的追随者之前一定要确定这样做确实有意义，而且不要仅仅因为其它的什么人如此自己也如此。

本书的最后一章讨论编写无错代码应持正确态度的重要性。如果没有正确的态度，发现错误和防止错误就好比在冬季大开着窗户给房间加热，虽然也能达到目的，但要浪费大量的能量。

本书除第 4 章和第 8 章以外都配有练习。但要注意，这些练习并不是要测验读者对相应内容的理解。实际上更多的是作者想在该章的正文中阐述却没能放进去的要点。其它的练习为的是让读者思考与该章内容有关的一些问题，打开思路，琢磨一下以前未曾考虑过的概念。

无论哪种情况，都是想通过练习再补充一些新的技巧和信息，因此值得一读。为使读者了解作者的意图，本书在附录 C 中提供了所有练习的答案。大部分章节还给出了一些课题，但这些课题没有答案，因为这种课题通常是任务，而不是问题。

规则或者建议

本书的编排类似于 Brian Kernighan 和 P. J. Plauger 所写的程序设计经典著作《The Elements of Programming Style》。该书出于 William Strunk Jr. 和 E. B. White 所写的重要经典著作《The Elements of Style》。这两本书采用同样的基本概念表达方法：

- 给出一个例子；
- 指出该例子中的某些问题所在；
- 用一般的准则改进该例子。

确实，这是个程式，而且是使读者感到舒服的程式，因此本书同样采用了这一程式。作者力图使本书读起来是一种享受，尽管它有着公式的性质。希望读者会觉得本书很有趣。

本书还给出一些似乎不应违背的“一般准则”。我们的第一条准则是：

每条准则都有例外

由于准则是用来说明一般情况的，所以本书一般并不指明准则的例外情况，而把它留给读者。我相信，当读者读到某个准则时，肯定会怀疑道：“噢，当……时，不是这样的……”。如果某个人对你说：“不能闯红灯”，虽然这是一条准则，但你肯定能够举出一种特殊情况，在这种情况下闯红灯倒是个正确的行动。这里关键是要记住准则只是在一般情况下才有意义，因此只有理由十分充足时，才可以违背准则。

关于本书代码的说明

本书的所有代码都是按 ANSI C 写的，并且通过了 MS-DOS、Microsoft Windows 和 Apple Macintosh 上五个流行编译程序的测试：

Microsoft C/C++ 7.0

Microsoft 公司

Turbo C/C++ 3.0

Borland 国际公司

Aztec 5.2

Manx 软件系统公司

MPW C 3.2

Apple 计算机公司

THINK C 5.0

Symantec 公司

还有一个问题：如果读者想从本书中摘取代码用在自己的程序中，那要小心。因为为了说明书中的论点，许多例子都有错误。另外，书中用到的函数虽然名字和功能都与 ANSI C 的标准库函数相同，但已对相应的界面进行了一些小的修改。例如 ANSI 版 `memchr` 函数的界面是：

```
void* memchr(const void* s, int c, size_t n);
```

这里 `memchr` 的内部将整数 `c` 当作 `unsigned char` 来处理。在本书的许多地方，读者都会看到字符类型被显式地声明为 `unsigned char`，而不是 `int`：

```
void* memchr(const void* pv, unsigned char ch, size_t size);
```

ANSI 标准将所有的字符变元都声明为 `int`，是为了保证其库函数同样可以用于 ANSI 标准之前编写的非原型程序，这时程序使用 `extern` 声明函数。由于在本书中只使用 ANSI C，所以不必考虑这些向下兼容的细节而可以用更加精确的类型声明以改进程序的清晰程度并用原型进行强类型检查（详见第 1 章）。

“提到 Macintosh 了吗？”

出于某种原因，一本书如果没有提到 PDP11，Honeywell 6000，当然还有 IBM 360，就不会被认真对待。因此，我在这里也提到了它们。仅此而已，读者在本书中再也不会见到这些字眼、读者见到最多的是 MS-DOS，Microsoft Windows，特别还有 Apple Macintosh，因为近年来我一直为这些系统编写代码。但是应该注意，本书中的任何代码都不受这些特定的系统约束。它们都是用通用的 C 编写的，应该能够工作于任何的 ANSI C 编译程序下。因此即使读者使用的系统本书没有提及，也不必担心这些操作系统的细节会产生障碍。

应该提到的是在大多数的微机系统中，用户都可以通过 `NULL` 指针进行读写，破坏栈框架并在内存甚至是其它应用程序的内存区中留下许多的无用信息，而硬件并没有什么反应，听任用户为所欲为。之所以提到这一点，是因为如果读者习惯于认为通过 `NULL` 指针进行写操作会引起硬件故障的话，那么可能会对本书中的某些语句感到迷惑不解。遗憾的是，目前微机上的保护型操作系统仍不普及，破坏内存的隐患必须通过硬件保护（通常它也不能提供充足的保护）之外的方法才能发现。

有错误就有错误

不必为本书的读者定义什么叫错误，相信读者都知道什么是错误。但是错误可以分为两类：一类是开发某一功能时产生的错误，另一类是在程序员认为该功能已经开发完成之后仍然遗留在代码中的错误。

例如在 Microsoft 中，每个产品都由一些绝不应该含有错误的原版源代码构成。当程序

员给产品增加新功能时，并不直接改变相应的原版源代码，改变的是其副本。只有在这些改变已经完成，并且程序员确信相应代码中已经没有错误时，才将其合并到原版源代码中。因此从产品质量看，在实现指定功能过程中不论产生多少个错误都没有关系，只要这些错误在相应代码被并入原版源代码之前被删除掉就行。

所有的错误都有害，但损害产品最危险的错误是已经进入原版源代码中的错误。因此，本书中提到的错误指的就是这些已经进入原版源代码中的错误。作者并不指望程序员在键入计算机之前总是写出没有错误的代码，但确信防止错误侵入原版源代码是完全可能的。尤其是程序员在使用了本书提供的秘诀之后，更是如此。