知名计算机先驱历经四十年考验的经典之作

“文津图书奖”译者高博最新译本

修炼编程功力、提高编程格调的终极宝典

the elements of programming STYLE

编程格调

[美] Brian W. Kernighan　P.J. Plauger　著

高博　徐章宁　译

人民邮电出版社

POSTS & TELECOM PRESS

编程格调

[美] Brian W.Kernighan 著
P.J.Plauger

高 博 徐章宁 译

人 民 邮 电 出 版 社

北 京

内 容 提 要

本书是编程惯用法和规则的实践指南。全书从表达、控制结构、程序结构、输入和输出、常见错误、效率和测试工具、方档等多个角度，概括了程序设计的最佳实践或规则，并通过代码示例加以分析和阐释。

本书两位作者都是程序设计领域的大师级人物。他们四十年前在本书中给出的 70 多条最佳实践和规划，大多数在今天仍然适用。

本书堪称计算机专业人士和程序员的必读的经典之作，适合于不同层级的程序员和计算机相关专业的学生参考阅读。

◆ 著 [美] Brian W.Kernighan P.J.Plauger
译 高 博 徐章宁
责任编辑 陈冀康
责任印制 张佳莹

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷有限公司印刷

◆ 开本：720×960 1/16
印张：12.25
字数：206 千字 2015 年 1 月第 1 版
印数：1-0000 册 2015 年 1 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-8414 号

定价：00.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315

版权声明

作者简介

Brian W. Kernighan，全球知名、德高望重的计算机先驱，在程序设计方法论和软件工程方面做了大量开创性的工作。他曾长期在贝尔实验室工作，现在普林斯顿大学计算机科学系任教。他著有数本经典教材，包括与 Dennis Ritchie 合著的传世之作 *The C Programming Language*、与 Rob Pike 合著的 *The Practice of Programming*，以及最近出版的科普图书 *D is for Digital* 等。他还是 AWK 编程语言的发明者，这种语言广泛地应用在 UNIX/Linux 应用中。“K&R C”和“AWK”中的“K”都代表 Kernighan。

P.J. Plauger，全球知名的计算机科学家、C/C++技术专家以及技术图书作者，更是数个标准 C/C++程序库的作者。他曾经在贝尔实验室工作，现在任美国 Dinkumware 公司总裁。他曾经担任 *C/C++ Users Journal* 高级编辑，也是 *The Standard C Library*、*Standard C: A Reference* 和 *The Standard Template Library* 等图书的作者。

译者简介

高博，1983 年生，毕业于上海交通大学。目前在互联网金融创业公司任首席产品官兼首席质量官，在信息科学和工程领域有近 15 年实践和研究经验。酷爱读书和写作，业余研究兴趣涉猎广泛。译著包括图灵奖作者高德纳的《研究之美》和布鲁克斯的《设计原本》，以及 Jolt 大奖作品《元素模式》等，出版翻译作品计近百万字。

徐章宁，1984 年生，就读于上海交通大学，硕士毕业后就职于 EMC 中国卓越研发集团，现任 EMC 公司高级系统管理工程师，从事软件运维工作多年，钟爱开源软件。对各类知识有广泛兴趣，平日喜爱参与问答网站讨论，热爱读书摄影和写作。

译者序

这本尘封了近 40 年的著名教材此次重新翻译，既是我们向前辈的致敬，也是我们重读经典的努力和尝试。译者第一次看到本书，是在 1998 年参加全国信息学竞赛期间，彼时也正是计算机工业在中国进入黄金发展期的前夜，读后的心潮澎湃历历如昨。此后，在和很多业界知名的前辈如 Donald E. Knuth、Bjarne Stroustrup 和 Anders Hejlsberg 等交流的过程中，也不止一次地听到本书的名字，可见本书影响之纵深。

为什么本书能有如此的影响力？也许会有人说，是因为作者的名气，但这是事后的见解。两位作者的功底固然是力透纸背，可放在当时的历史背景下，其实主要的原因却在于，它开创了一个程序设计习惯用法（idiom）的先河。和两位作者此后的作品相比，本书中很多地方反而可以看出，他们当时还是以新锐作家的口吻，在创立自己的观点，而非早已站在制高点处向大众布道而已。这是本书最为宝贵之处，我们从中可得以窥见现在已经几乎成为常识的观点，发端于怎样的初心，并经历过怎样的磨砺，才逐渐站稳了脚跟。也正是因为如此，本书对于当时的软件业界而言，不啻是黑暗中的一道闪电，照亮了人们前行的步伐，所以才能在这么多高手的心中留下难以磨灭的印象。

而对于今天中国的软件业界，本书的重新翻译和出版，同样意义重大。求新、求异，却不重视程序设计基本功和工程素养的风气，近年来有令人不安的抬头趋势。我们思考再三，决定将本书的译名定为《编程格调》，也是为了强调书中反复申明、历久弥坚的习惯用法以及它们反映出来的宝贵思想，对于当前现实的指导意义。阅读本书，可以回答一个非常基本的问题：怎样去构建软件、构建怎样的软件，才能说成是“有格调”的，才能走得稳、走得远。

当然，对于一本近 40 年前的出版物，我们不可能要求它尽善尽美。比如书中采用的编程语言，在今天看来已经不是非常流行。但一个合格的程序员，应该可以比较迅速地掌握一门新语言的结构，至少看懂一段代码所要表达的含义。正如我国著名画家齐白石所言：“学我者生，似我者死”。我们要学会学习，吸收书中观点的精华，并和自己当下的工程实践相结合，而非邯郸学步。

感谢人民邮电出版社的陈冀康编辑为促成本书的出版而作出的大量努力，EMC 中国卓越研发集团的徐章宁高级工程师和本人一起完成了本书的重新翻译工作，这里也要向他的辛勤工作表达由衷的感谢。本书成稿的过程中，上海交通大学计算科学与技术系的张尧弼和梁阿磊教授、赛门铁克中国区技术负责人汤瑞欣总监、华为存储事业部严华兵高级经理和开源社区事业部杜玉杰工程师，以及 GitCafé CEO 姚欣宇等都参加过审阅并提出了宝贵意见。当然，由于译者能力所限，本书中仍然不可避免地存在更多的纰漏，这些理应由我们负责。我也想借此机会向在工作和生活上给了我莫大支持的父母和爱妻沈靓表达我内心最深处的敬意，希望本书的出版能给你们带来快乐。

2015 年元旦

于杭州苏堤

再版序言

自《编程格调》初版面世以来，计算机程序设计实践已经今非昔比。关于编程格调的讨论已登大雅之堂。历经“代码写过算数”的漫长岁月，学生、教师，以及计算界专业人士已经意识到程序可读性的重要。结构化程序设计已经作为一种极有价值的编程原理被普遍接受，而且人们也逐渐意识到，设计乃是一个重要的编程阶段，而这在过去经常是被忽视的。

我们对《编程格调》进行了一番彻底改写。在初版中，我们避免在任何地方直接使用“结构化程序设计”，以远离当时甚嚣尘上的宗派之争。而现在，人们的热情已经消退，我们反而感觉是时候讨论已在实践中经受住考验的结构化程序设计话题了。

本书再版时，加入了新的一整章来讨论程序结构，阐明如何运行自顶向下的设计写出结构良好的程序。全书都在讨论设计中遇到的种种问题。我们也谨慎地运用伪代码作为一种程序开发工具。

我们也重写了初版中的大量示例，以反映出（至少我们希望如此）对于如何编好程序的更深入理解。同样，书中还引入了不少新示例。初版中不少拿来作为榜样的示例，现在却成了反面教材。再版还加入了一些新的习题。最后，我们扩展、深化了作为编程格调的种种规则。

对于慷慨授权我们在书中重印他们教材中程序的作者和出版商，我们再次感觉受益匪浅。回过头来看我们自己写的程序就能明白，撰写出优秀的程序有着怎样高的要求。

我们同时要感谢阅读再版草稿的朋友，尤其要感谢 Al Aho、Jim Blue、Stu Feldman、Paul Kernighan、Doug McIlroy、Ralph Muha 和 Dick Wexelbalt，他们曾提出过非常有价值的建议。

Brian W.Kernighan
P. J. Plauger

初版序言

仅仅通过听取一般原理，就想很好地掌握程序设计，这是不可能的事。想要学好程序设计，就要一次又一次地观察体会实际的程序是如何通过运用一些久经实践考验的原理和常识，方得持续改进的。程序审阅能够教会人们如何重写程序，再进一步教会人们如何写出更好的程序。

本书就是建立在对大量“实际”程序的研究之上的成果，其中的每个程序都有着一种或数种格调方面的毛病。我们讨论了每个例子存在的缺点，并采用更好的方式将其重写，然后由这个特殊情况推出一般原理。我们的出发点是实用的、接地气的。我们关心的是要改进当前程序设计的工作，而不是想要创建一套繁杂理论来告诉人们如何进行程序设计。所以，本书可以用作任何水平的程序设计课程的补充材料，也可以让已有经验的程序员们读后获得温故而知新的效果。

书中的所有示例皆以 Fortran 和 PL/Ⅰ写就，因为这两种语言不仅被广泛采用，而且互相类似，只要会看其中一种，则另一种也不言自明（无论采用哪一种语言来写程序，我们都避免采用复杂结构，遇到不可避免的习惯用法时，也会加以解释）。而有关编程格调的原理，则适用于一切语言，包括汇编码。

我们的目标在于仅运用少量的篇幅即能讲授编程格调的妙处，所以我们有一说一。贯穿全书的是一系列规则，这样方能彰明我们习得的教训。每一章都以一段总结和一组“思考要点”收尾，其中会给出一些练习，并使得读者有机会继续考查正文中没有讲全的一些议题。我们在书的末尾将规则集中列出，以飨读者。

这里特别要讲一讲本书中程序实例的来源，它们全部选自其他的程序设计教科书。换言之，我们不通过自己撰写程序来论证我们的观点，而是使用那些已经完成的产品，即那些由富有经验的程序员撰写并且发布的程序。这些程序往往就是新程序员们入行后见到的首批程序，本来我们希望这些例子能起到编程格调的

示范作用，遗憾的是却常常事与愿违。这些教科书中举的例子往往本想说明自己在某个方面做得出色，结果却适得其反，暴露了问题。（我们不会故意使坏，通过断章取义来使程序在该语境下显得写得不好。）

我们还要特意声明：我们不是要以个人或集体名义批判这些教科书的作者们。缺点的存在仅仅表明我们不过都是凡人，在编程或写书这样压力巨大的脑力劳动中，有些事情做得不完美诚属难免。毫无疑问地，本书中所提到的一些“好”程序，一定也会被未来的某位作者举作“坏”程序的反面例子。我们也诚心地希望未来的这位作者及其读者也能通过仔细研究这些程序的经验而学到东西。

一本编程格调手册的问世，离不开很多人开创性的工作，他们中有相当数量的人就写过非常出色的教科书。比如，D. D. McCracken 和 G. M. Weinberg 很久以来就在教学中呼吁简单性和明晰性的极端重要。E. W. Dijkstra 和 Harlan Mills 在结构化程序设计方面取得的成果，帮助我们明确了指定控制流的相关规则。本书的篇章安排和语言风格都深受 W. Strunk 和 E. B. White 的名著 *The Elements of Style* 一书的影响，我们通过集中精力于格调的本质和实用方面，来达到该书的简明程度。

我们受益于太多人的帮助和鼓励，特别是授权我们复用在本书正文中的程序作者和出版社，这样的合作极其难能可贵。

我们在贝尔实验室的朋友和同事们提出过大量有用的建议，正是由于这些建议，我们得以在出版前纠正了不少本来可能贻笑大方的谬误。特别要感谢 V. A. Vyssotsky 与我们一起进行了数次修订工作，对于他富有洞见的评论和对本书成形过程中每一阶段的热情支持，我们深表谢意（还有，我们在书中“无耻盗用”了他的几句格言）。我们还想感谢 A.V. Aho、M. E. Lesk、M. D. Mcllroy 和 J. S. Thompson 给予本书的时间投入和莫大帮助。

我们之所以能把手稿直接录入 PDP 11/45、编辑源码、校验程序，并对最终版文本排版，所有这一切都要仰仗一个独一无二的、灵活的操作系统，这就是 UNIX。K. L. Thompson 和 D. M. Ritchie 是 UNIX 的首席架构师，除了帮我们校稿以外，他们还在我们写书时帮我们最大程度地利用好 UNIX 系统。J. E. Ossanna 编写了排版程序并修改了数处，以满足我们的特殊要求。在此致谢。

Brian W. Kernighan

P. J. Plauger

目录

第1章 绪论……1

撰写简洁的程序——不要耍小聪明……2
第2章 表达……13

简单并且直接地表达你要说的意思……14
使用库函数……14
避免使用临时变量……16
代码要清晰，不要为了“效率”牺牲可读性……16
让机器干脏活……17
用函数调用替代重复的表达式……18
加括号来避免歧义……21
选择不会被混淆的变量名……21
避免使用 Fortran 的算术 IF……23
避免不必要的分支……25
使用语言好的特性，避免使用不好的特性……25
不要使用条件分支来代替一个逻辑表达式……26
用“电话测试”来检查可读性……29
第3章 控制结构……39

使用 DO-END 和缩进来界定语句组……40
用 IF-ELSE 强调两个操作中只有一个被执行……42
用 DO 和 DO-WHILE 来强调循环的存在……45

确保你的程序是自顶向下阅读的……46
使用 IF…ELSE IF…ELSE IF…ELSE 来实现多路分支……47
使用基本的控制流结构……48
先用容易理解的伪语言编写代码，然后再翻译成 你需要使用的语言 ……52
避免使用 THEN-IF 和空 ELSE……55
避免使用 ELSE GOTO 和 ELSE RETURN……56
判断要尽可能紧挨着与之相关的操作……58
使用数组来避免重复的控制流……61
选择可以简化程序的数据表示方法……63
不要止步于第一遍的代码草稿……66
第 4 章 程序结构……71

模块化，使用子例程……74
让模块之间的耦合变得可见……75
每一个模块都应该做好一件事……76
确保每一个模块都隐藏好一些东西……78
以数据为导向来构建程序的结构……80
不要修补烂代码——重写它……84
分块编写和测试大的程序……91
对于递归定义的数据结构使用递归过程……91
第 5 章 输入和输出……97

校验输入的合法性和合理性……100
保证输入数据不会违背程序的限制……101
利用文件结束符号或结束标志来终止输入， 不要让用户去计数……102
识别出非法输入数据，如果可能则纠正之……103
使用统一的形式处理文件结束条件……105
让输入数据易于准备，并让输出数据意义不言自明……108
使用统一的输入格式……110
让输入数据易于校对……111

尽可能选择自由格式输入……112
使用含义自明的输入，指定默认值，将以上二者都输出……112
将输入与输出局限在子例程中……116
第6章 常见错误……119

确保所有的变量在使用之前都被初始化……120
不要停留在一个 bug 上……122
使用调试编译器……124
用 DATA 语句或 INITIAL 属性初始化常量， 用可执行语句初始化变量……125
小心“差一”错误……126
要注意对不等式进行正确的分支……126
避免循环有多个出口……128
确保你的代码巧妙地“不做事情”……131
在边界值上测试程序……135
预防性编程……136
10.0 乘以 0.1 不等于 1.0……137
不要比较浮点数是否相等……139
第7章 效率和测试工具……145

先做对，再做快……147
在提高程序运行速度时，要保持其正确性……149
先把程序改得更简洁，再提高其运行速度……150
不要为了“效率”上的蝇头小利而牺牲程序的简洁性……151
让编译器执行平凡优化……151
不要勉强地复用代码，应该进行改编……152
保证特殊情况真的有特殊性……155
保持简单性，反而会更快……157
不要为了提高速度而画蛇添足——去寻找更好的算法……159
在程序中放置测试语句，“增效”之前先执行测算……161
第8章 文档……165

确保注释和代码一致……167
不要用注释复述代码做的事情， 每个注释都要有实际意义……167
不要注释糟糕的代码——重写它……169
使用含有意义的变量名……170
使用含有意义的语句标签……171
程序的格式要有助于读者的理解……171
用缩进来体现程序的逻辑结构……172
记录你的数据规划……175
不要过度注释……176
结束语……180

第 1 章　绪论

考查以下程序片断。

```
   DO 14 I=1,N
   DO 14 J=1,N
14 V(I,J)=(I/J)*(J/I)
```

只要对 Fortran 语言有一定的了解，我们就知道，这个双重嵌套的 DO 循环是要对具有 N×N 个元素的矩阵 V 的各个元素赋以某值。怎样的值呢？I 和 J 是正的整型变量，在 Fortran 语言中，整型除法会向零取整。因此，当 I 小于 J 时，（I/J）为 0。反之，当 J 小于 I 时，（J/I）为 0。而当 I 等于 J 时，两个因数皆为 1，所以（I/J）*（J/I）等于 1，当且仅当 I 等于 J，否则皆为 0。这段程序意在将 V 的对角元素置 1，而其余元素置 0（从而 V 成为一单位矩阵）。多么聪明啊！

果然如此吗？

设想，你是在一个较大的程序中遭遇了这段程序。如果你对 Fortran 语言造诣颇深，那你可能会很欣赏此处对整型除法的妙用。但也有可能你会觉得吃惊不小，因为明明有更简单的做法，这里采用的做法却要做两次除法、一次乘法和一次从

整型到浮点型的强制转换。更大的可能是，你不得不重复上面的推理过程。极大的可能是，你只是有了一个囫囵的印象，知道它向数组中放置了某些有用的东西。只有当你有了强烈的动机，比如要对程序进行调试或改动时，才会回过来猜测它的确切含义。

这段程序这样写会更好。

```
C  MAKE V AN IDENTITY MATRIX
      DO 14 I = 1,N
         DO 12 J = 1,N
12          V(I,J) = 0.0
14       V(I,I) = 1.0
```

先将各行置0，然后将对角元素置1。现在含义十分明确，代码运行得也更快了。如果我们是用PL/Ⅰ撰写程序的，就可以表达得更明显。

```
/* MAKE V AN IDENTITY MATRIX */
      V = 0.0;
      DO I = 1 TO N;
         V(I,I) = 1.0;
      END;
```

无论哪种情况，要点在于要使代码的意图不会被误解，而非炫技。在这里，存储空间和执行时间相对而言都不重要，因为建立单位矩阵一定是整个程序中的一小部分。难懂代码的问题在于它难以调试和修改，而这些已经成为计算机程序设计中最困难的问题。此外，过分聪明的程序会增加与原意发生偏离的危险。

撰写简洁的程序——不要耍小聪明

暂停一下，让我们反思一下到目前为止我们都做了些什么。我们研究了一段程序，该程序原样引自某本程序设计教材。我们讨论这段程序的优点和缺点，然后着手对其加以改进。（未必达到完美——仅仅是改进。）在分析和改进中，我们总结出一条规则或是一般结论，该规则听起来不免有点像是有着纯粹抽象的普适性，但它合情合理，在遇到具体情况时就能具体运用。

本书其余部分大抵如此，即从教材中选取实例，尔后是讨论和改进，再得出

规则，如此反复。读完本书之后，你应当学会评判自己的代码。更重要的是，你应该具备了一开始就把程序写得更好的能力，从而不会十分招惹非议。

我们尝试按逻辑由浅入深的顺序对例子进行分类，但是正如你所见，实际程序如同文学习作，经常同时违反若干条良好实践的规则。因此，我们的分类不免有时会显得有些任意，而且有时也不得不话分两头。

虽然大部分的例子都比刚才的那个要长，但也不会长得过头。即使是初学者，通过讨论，也可以跟得上。实际上，大部分长程序都会在你眼皮底下施以缩身术。完全正好的尺寸往往只是幻想，它只不过体现了对于程序存在改进的要求。

所有的例子皆以 Fortran 和 PL/Ⅰ写就，但是你只熟悉其中一种语言，甚至一种也不熟悉，你所能遭遇的困难都不会超过程序本身的尺寸。打个比方，你可能不会用 PL/Ⅰ编程，但你肯定能读懂 PL/Ⅰ程序，并理解我们讨论的要点，边读边学会让你更轻松地掌握 PL/Ⅰ。

例如，这里就是一小段 PL/Ⅰ程序，我们将在第 4 章中详细讨论整个程序。

```
          IF CTR > 45 THEN GO TO OVFLO;
        ELSE GO TO RDCARD;
OVFLO:
      ...
```

第一个 GOTO 和第二个 GOTO 纠缠在一起，看起来有点乱。用“<=”代替“>”，就可以改写如下。

```
      IF CTR <= 45 THEN GOTO RDCARD;
OVFLO:
      ...
```

语句少了一个，逻辑变简单了，OVFLO 也变得多余了。教益是什么呢？分支和分支不要纠缠在一起，把关系检测表达式调个方向，程序就更容易理解了。下面我们看一个有着相同问题的 Fortran 例子，这么一来就可以得出一个重要的结论，语言细节虽然各自不同，但编程格调的原理则是一致的。分支与分支纠缠在一起的话，在任何语言中都会引发困扰。哪怕你是用 COBOL、BASIC、汇编或其

他语言，你在此处习得的原理都可以应用。

上例看起来有小题大作之嫌。无论如何，原始的程序含义还算是很清楚的。问题在于，一个小问题也许不会造成大的破坏，但是多处引发困扰语句形成的积累效应，就会使程序十分费解了。

下面看个稍长些的例子。

以下是个求某数（X）平方根（B）的典型程序：

```
  READ(5,1)X
1 FORMAT(F10.5)
  A=X/2
2 B=(X/A+A)/2
  C=B-A
  IF(C.LT.0)C=-C
  IF(C.LT.10.E-6)GOTO 3
  A=B
  GOTO 2
3 WRITE(6,1)B
  STOP
  END
```

由于程序稍长，我们分成若干层次进行研究。比如，在进入细节之前，先想想这个程序是不是那么“典型”。作为一个求平方根的子例程，不大可能做成一个主程序的样子，还需要自行读入数据——带有一个自变量的函数就非常够了。即使我们真的需要一个主程序来计算平方根，又有多大可能会希望它在停止运行前只算出来一个平方根呢？

这种将程序限制过死的趋势，提醒了我们应该怎样编写想用于通用目的的程序。不久我们就将遇到只跟踪不多不少 17 位销售的程序，只对正好 500 个数排序的程序，以及只走一个迷宫的程序。我们猜想，这样把程序改写为通过编译器获得参数的场合，是非常多的。

让我们继续看这个平方根程序。它的实现采用了牛顿法，这的确是许多求平方根库例程的核心（但我们不必精确地了解它如何运作）。给以适当的数据，这个方法很快就会收敛。不过，如果 X 是个负数，此程序就会陷入死循环。（不信可以一试。）好的程序应当能够返回错误信息或诊断消息。当 X 为 0 时，这个程序

会在行号为 2 的语句处算个不休，这种情况应该分开处理。0 的平方根就给一个结果 0 即可。

即使 *X* 取严格的正值，这一程序也可能算出无意义的结果。问题出在收敛校验的运用上。

```
C=B-A
IF(C.LT.0)C=-C
IF(C.LT.'0.E-6)GOTO 3
```

应该利用 Fortran 语言特性，把第二行改成：

```
C = ABS(C)
```

为避免有人把 10.E-6，即“10 的-6 次幂”看错，第三行中的常数应改成 1.0E-5，甚至改成 0.00001 都行。其实，为了直抒胸臆，不要绕弯子，这头 3 行应该直接写成：

```
IF (ABS(B-A) .LT. 1.0E-5) GOTO 3
```

现在，这个校验的含义就非常清楚了。但可惜它有错。

若 *X* 充分大，两个尝试平方根的绝对差值就很有可能永远不小于 1.0E-5 这个随心所欲选择的收敛阈值，除非这个绝对差值严格为零。因为，计算机表示的数字精度有限。此绝对差值是否总可以收敛到 0，属于数值分析上的一个尚无定论的问题。如果 *X* 取较小值，则这个标准早早地就会被满足，此时还远未得到一个理想的估值。如果我们采用相对于原始数据的趋近估计，而非绝对收敛作为标准，则对于大多数的正自变量，均可获得 5 位有效数字。

```
C COMPUTE SQUARE ROOTS BY NEWTON'S METHOD
 100  READ(5,110) X
 110     FORMAT(F10.0)
C
      IF (X .LT. 0.0) WRITE(6,120) X
 120     FORMAT(1X, 'SQRT(', 1PE12.4, ') UNDEFINED')
C
      IF (X .EQ. 0.0) WRITE(6,130) X, X
 130     FORMAT(1X, 'SQRT(', 1PE12.4, ') = ', 1PE12.4)
C
```

```
      IF (X .LE. 0.0) GOTO 100
      B = X/2.0
  200 IF (ABS(X/B - B) .LT. 1.0E-5 * B) GOTO 300
      B = (X/B + B) / 2.0
      GOTO 200
  300 WRITE(6,130) X, B
      GOTO 100
      END
```

修改过的程序还够不上说是求平方根的典型例程的标准，我们也不打算深入浮点数计算的技术细节，来完成这个例程。可是，这一程序的通用性倒是足够典型，评判和改写后它变得好多了。

再举本章的最后一例，通过研究我们会发现它有若干缺点，以下是个排序例程。

```
   DIMENSION N(500)
   WRITE (6,6)
 6 FORMAT (1H1,26HNUMBERS IN ALGEBRAIC ORDER)
   DO 8 I=1,500
 8 READ (5,7) N(I)
 7 FORMAT (I4)
   DO 10 K=1,1999
   J=K-1000
   DO 10 I-1,500
   IF(N(I)-J)10,9,10
10 CONTINUE
   STOP
 9 WRITE (6,95) N(I)
95 FORMAT (1H ,I4)
   GO TO 10
   END
```

程序不但缺乏通用性，而且算法漏洞百出，代码写得不明不白，还有一个印刷错误。其中

```
DO 10 I-1,500
```

这一行中的“-”应该是个“=”。这个程序想方设法要达成的目的之一，在于展示 DO 循环体可以通过向外跳转，然后跳转回来的方法进行扩展的特性，不过在本例中，DO 循环体外加扩展的语句能够一起写到一行里。

```
   DO 10 I = 1, 500
      IF (N(I) .EQ. J) WRITE (6,95) N(I)
95       FORMAT(1X, I4)
10 CONTINUE
```

关于这一点，一个更重要的问题在于，从根本上说，是否应该鼓励程序员去

使用扩展范围。在计算机程序中非必要的跳转，已经被证明是错误的一大来源，并且通常这表明程序员没能很好地掌控代码走向。本例中杂乱无章的语句行号，往往也是思想混乱的外在表现。

此程序还有其他的毛病。它读入 500 个数，每张卡片一个，然后简直是以最低可能的效率对这些数排序，方法是将每个数与−999 到+999 之间的所有整数进行比较。它只做一次，只处理一组数，然后就停止了。

等等，好像哪里不对。由于指定了 I4 输入格式，在正号可能被忽略的前提下，可能读入的最大正数是 9999，而这个程序从一开始的设计来说，就无法列出 4 位数。要纠正这个问题，算法的耗时就得增加 5 倍都不止，而通用性却一点儿都没有增加。如果套用这种方法来扩展程序以处理更大的整数，速度会减慢若干数量级。要让它处理浮点数，更是想都不要想了。

由于我们从根本上不赞同这个程序的思路，这里就不重写它了。（第 7 章中列举了几个更好的排序程序。）我们只想表明，同一程序可从不同的视角来看。批判性的阅读不要止于发现一个键入错误，甚至不良的整体思想。在后面各章中，我们将探讨以上看到的，以及其他的问题，这些都会极大地影响到编程格调。

第 2 章开篇，我们会研究如何将单个语句写得简洁。写好算术表达式和条件语句（IF），通常是计算机程序设计的基础课中首先要讲的内容。在深入其他语言特性之前，先把这些基础牢牢掌握。

第 3 章讨论程序的控制流结构，即怎样通过循环语句和决策语句来控制程序流向。同时，这章还说明了如何表示数据能使程序设计工作变得尽可能轻松，以及采用何种数据结构可以获得清晰的控制流。第 4 章的主题是程序结构，讨论了如何将一个程序分解成易于处理的块。这两章花费了不少笔墨在正确使用结构化程序设计及坚实的设计技术上。

第 5 章考查输入和输出，包括如何在撰写程序的过程中，使其免受差劲的输入数据伤害，以及如何组织输出以使程序在运行中获取尽可能多的收益。第 6 章研究了许多常见错误，并讨论了如何发现和改正它们。

和通常的做法相反，效率和文档在最后两章才出现。虽然这两个主题都很重要，也很值得学习。但我们认为对它们的重视程度有点过头了，尤其是在很多入门课程中存在这个不良的偏重，而程序简洁性等格调却因此没有得到应有的关注。

以下是我们在评判程序时一些通行规则的说明。

（1）程序尽量以本来面目示人，只要打印系统支持。格式、印刷错误和词法错误也照单全收。（例外情况是，有 3 个 PL/Ⅰ程序的表示方法从 48 个字符改成了 60 个字符。）

（2）我们的常规做法是从程序中摘录片断，这样可以集中精力在其本质方面。我们相信，所讨论的缺点是代码内固有的，并非由于断章取义而造成或加重了问题。我们尽可能在摘录时不失语境，也在想方设法从根本上解决程序的原始版本中想要解决的问题，只有这样对比才能做到公平，即使有时这样做会使得我们无法穷尽程序的改善空间。

（3）我们不会因为例子使用了非标准语言特性（例如，Fortran 语言中的混合模式算术[1]）而对其吹毛求疵，除非这种用法很不常见或非常危险。大多数编译器都能接受非标准结构，而标准本身也是与时俱进的。不过还是要牢记，不常用的特性难以移植，而环境一旦变化，它们是首当其冲出现状况的。

我们自己使用的 Fortran 语言严格遵循 1966 年的 ANSI 标准，除了用引号包含的 Hollerith 字符串（我们拒绝进行字符计数）。所用 PL/Ⅰ符合 IBM 最新编译器版本 1、发行版 3.0 的标准。虽然我们也见过新版的 Fortran 和 PL/Ⅰ版本，它们使更好的程序设计成为可能，但尚未得到广泛应用，所以我们并未使用它们来编写任何示例。

（4）在针对数值算法的讨论中（例如，前面的平方根程序），我们不打算诊疗其全部症状。诸如溢出、有效精度丢失，以及其他一些数值方面的小错的对策，

[1] 指 Fortran 语言中，在算术表达式中混用整型和浮点型。一般规定，操作数中只要有一个是浮点型，则结果为浮点型，否则为整型。

超出了本书的范围。但我们坚持认为，最基本的预防措施还是要有的，比如用相对差值校验代替绝对差值校验，以及避免被零除等，为的是保证只要给定合理的输入，就可以收获正确的计算结果。

（5）本书中的每行代码都通过了编译，直接利用程序文本即可被计算机正确读取。所有程序均经过测试（Fortran 程序的测试机是一台 Honeywell 6070，PL/Ⅰ程序的测试机是一台 IBM370/168）。Fortran 程序经过了一个校验器的校验，以监督它们是否符合 ANSI 标准。

即便如此，错误还是在所难免。我们鼓励读者抱着怀疑的态度，来看待我们的每句看上去不太顺眼的话。请动手做校验，彻底检查一番。不要把计算机的输出结果奉为圭臬。假如你学会了谨慎对待别人的程序，你也就能够更好地审视自己的程序了。

思 考 要 点

1.1 $n \times n$ 的矩阵有 n^2 个元素，因此，初始化这么一个矩阵就需要 n^2 个赋值语句。采用经典方法完成两个 $n \times n$ 矩阵的乘法，或求解由 n 个 n 元方程所构成的方程组，要做 n^3 阶运算（这些都是矩阵处理程序的日常工作）。试提出相关论据以支持下列猜想。

如果 $n \geq 10$，初始化矩阵所花费的时间无关紧要。

如果 $n < 10$，初始化矩阵所花费的时间无关紧要（提示：输入和输出时执行的转换比算术费时要多）。

1.2 在本书的初版中，我们编写过求平方根的例程如下。

```
C             COMPUTE SQUARE ROOTS BY NEWTON'S METHOD
   10 READ(5,11) X
   11 FORMAT (F10.0)
      IF (X .GE. 0.0) GOTO 20
         WRITE(6,13) X
   13    FORMAT (' SQRT(', 1PE12.5, ') UNDEFINED')
         GOTO 10
   20 IF (X .GT. 0.0) GOTO 30
         B = 0.0
         GOTO 50
   30 B = 1.0
   40    A = B
         B = (X/A + A)/2.0
         IF (ABS((X/B)/B - 1.0) .GE. 1.0E-5) GOTO 40
   50 WRITE(6,51) X, B
   51 FORMAT(' SQRT(', 1PE12.5, ') = ', 1PE12.5)
      GOTO 10
      END
```

这个程序更“高效”，因为它没有重复校检。你喜欢哪个版本？为什么？这个改变造成多少时间和空间差异？这两者都彰明了 Fortran 语言的什么不足之处？

1.3 在求平方根的例程中，我们看到校验时采用 1.0E-5 这一绝对阈值作为收敛标准很危险，所以建议采用某种相对标准代替。如何把下列函数用到我们的例程中？

```
REAL FUNCTION RELDIF(X, Y)
RELDIF = ABS(X - Y) / AMAX1(ABS(X), ABS(Y))
RETURN
END
```

AMAX1 是个 Fortran 函数，它返回两个或两个以上浮点数中的最大值。上述函数在求平方根的例程中可能遇到的所有值都能得出正确结果。但是如果是在更一般的场合下，有没有什么 X 和 Y 的值会让这个函数出错？

结束语

是时候做一次盘点了。虽然我们已经在前面 8 章中谈及编程的方方面面，可还是有太多内容留在不言中。有些是限于篇幅不能展开，但大部分的省略属于故意留白。在语言、算法和数值方法领域，有许多好书可供在编程方面学有余力者参考。而本书的目标并非教授语言或算法，而是如何更好地编程。

低估编程格调的倾向，在程序员中普遍存在。我们常常过分乐观地认为：只要能拼凑出一段程序即可，无论过程有多么随性，它就能一次运行正确，次次运行正确。为什么要浪费时间去理顺几乎肯定正确的东西呢？更何况，编成的程序也许只会使用几个星期而已。

严格来讲，这个问题有两部分答案。第一部分是由“几乎”一词阐发而来。一段东拼西凑而成的垃圾程序很难处理。第一次编程时的自律使你写出正确程序的机会更大，即使错了也更容易改正确。如果一名程序员拿起第一稿程序就直奔代码面板或终端，那么比起他的细心同事来说，他在重写和调试程序上所花的时间会多得多。

第二部分是针对“只会使用几个星期”这一点。说到底，编程的差异来源于为了想要利用它的不同场合。但计算中心却存在大量的程序，虽是为了一时之需，但实际上却由于各种各样的压力服役了多年。有的时候，还不仅仅是压力，简直是暴力和扭曲了。为着新的应用而修改已有的程序，无论写得多差，往往也比一切推倒重来要简单。大型程序如操作系统、编译器，以及主流应用程序等，肯定不是用毕即抛的一次性程序。它们会不断地修改和演化。大多数专业程序员会把大量时间花在修改自己的和他人的程序上。再强调一遍，简洁

的程序才易于维护。

撰写难以识读的程序有一个借口：这不过是件私事，只有该程序的原作者才有机会再对它看上一眼。既然他已经在脑子里掌握了一切，那么也就不必一一记下。这的确是个貌似非常有理的论点，在非专业程序员中更是广为流传。但这就好比你在一张购物清单写下“夸脱牛奶、鱼、大盒”，而不是写下一个完整的句子。当然，如果这张购物清单要是写给别人看，你肯定得指明要买的是什么鱼，以及要买的是大盒装的什么东西。哪怕你只想让自己明白，但是在一年之后还能明白，你也得写下一个完整的句子。所以你可能会在日记中写下这样的话：“今天我去了超市，买了一夸脱牛奶、一磅大比目鱼和一份大盒装葡萄干。”

你撰写程序时，要做到仿佛是给别人看一样，因为一年以后你也就成了“别人”。在学校里人们只会学习如何写作文，而不是如何写购物清单。因为只要掌握了前者，后者就好办了。但是论及计算机编程，许多程序员似乎反过来认为只要掌握了如何写购物清单这样的三招六式，就足以应付撰写大型程序所需要的真功夫。实际上，差得还远呢。

我们所想要传达的思想精髓，就融汇在难以言传的“格调”一词之中了。与其说它是一组规则，还不如说它是一种方法或一种态度。所谓“编程高手”，其实就是掌握了一些规则，能够保持良好的编程格调。他们中的许多人都会阅读本书，并且感觉到共鸣。然而，如果你现在还在成为“编程高手”的学习途中，那么很有可能在阅读过程中你会忽视掉某些我们视为良好编程格调的内容。

编程格调
the elements of programming STYLE

本书堪称计算机专业人士和程序员必读的经典之作。全书从表达、控制结构、程序结构、输入和输出、常见错误、效率和测试工具、文档等多个角度，概括了程序设计中的最佳实践或规则，并通过代码示例加以分析和阐释。

作者四十年前在本书中给出的70多条最佳实践和规则，大多数在今天仍然适用。一方面，很多程序员通过遵循这些最佳实践和规则编写出可靠而稳定的程序；另一方面，一些调试人员甚至将本书作为调试程序的参考书和检查表。

正因为如此，本书的价值没有因为时间的推移而消磨，相反得到了更多的验证和凸显。时至今日，本书仍然被奉为程序员必读的经典之作。

本书中包含的一些重要的规则（我们称之为“格调建议”）如下：

- 撰写简洁的程序——不要耍小聪明；
- 代码要清晰，不要为了“效率”牺牲可读性；
- 使用语言好的特性，避免使用不好的特性；
- 先用容易理解的伪语言编写代码，然后再翻译成你需要使用的语言；
- 分块编写和测试大的程序；
- 校验输入的合法性和合理性；
- 在边界值上测试程序；
- 先做对，再做快；
- 保持简单性，反而会更快；
- 不要过度注释。

ISBN 978-7-115-37952-8

美术编辑：董志桢

分类建议：计算机／程序设计
人民邮电出版社网址：www.ptpress.com.cn