HZ BOOKS

PEARSON Addison Wesley

Modern C++ Design C++ 设计新思维

Exceptional C++ Style

40 New Engineering Puzzles, Programming Problems, and Solutions

（英文版）

（美） Herb Sutter 著

机械工业出版社
China Machine Press

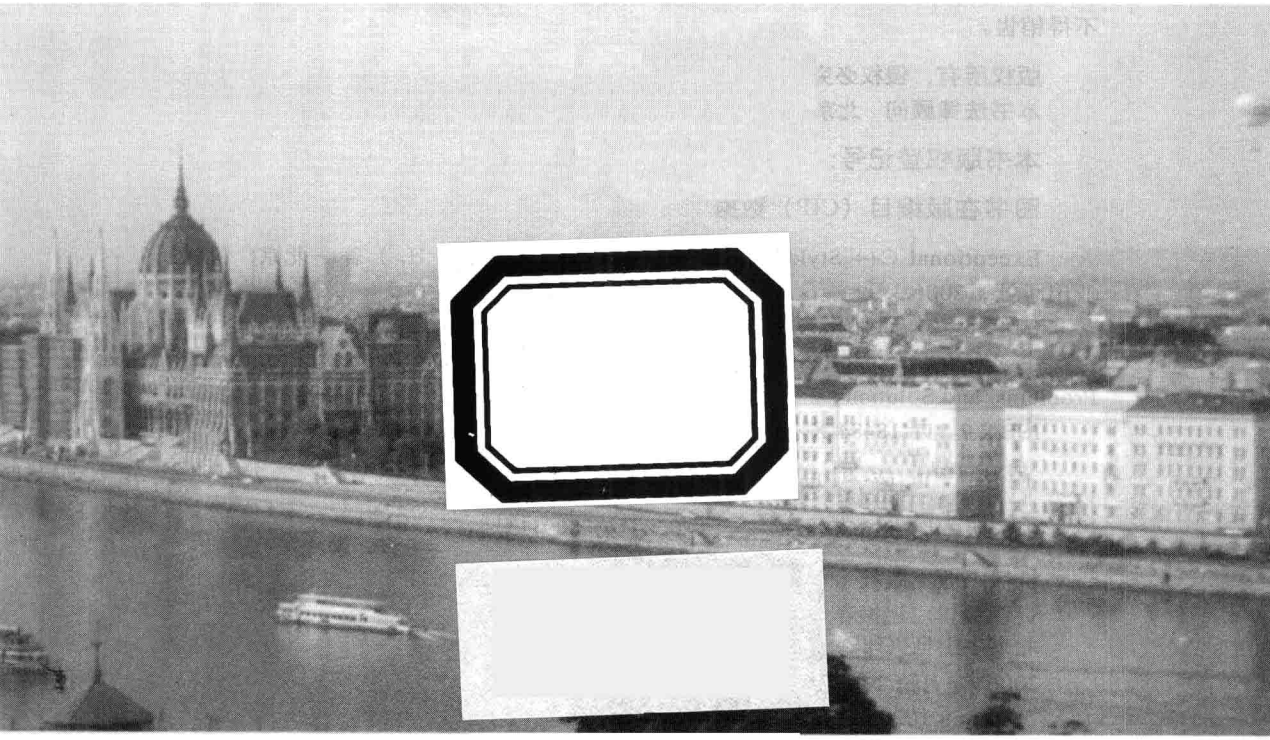
Modern C++ Design
C++
设计新思维

Exceptional C++ Style

（英文版）

40 New Engineering Puzzles, Programming Problems, and Solutions

（美） Herb Sutter 著



机械工业出版社
China Machine Press

本书版权登记号：图字：01-2006-0529

图书在版编目（CIP）数据

Exceptional C++ Style（英文版）/（美）萨特（Sutter, H.）著. - 北京：机械工业出版社，2006.3

（C++设计新思维）

书名原文：Exceptional C++ Style: 40 New Engineering Puzzles, Programming Problems, and Solutions

ISBN 7-111-18484-X

Ⅰ. E…　Ⅱ. 萨…　Ⅲ. C语言 - 程序设计 - 英文　Ⅳ. TP312

中国版本图书馆CIP数据核字（2006）第008911号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2006年3月第 1 版第 1 次印刷

718mm × 1020mm 1/16 · 21.75印张

印数：0 001-3 000册

定价：45.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

“C++设计新思维”丛书前言

自C++诞生尤其是ISO/ANSI C++标准问世以来，以Bjarne Stroustrup为首的C++社群领袖一直不遗余力地倡导采用“新风格”教学和使用C++。事实证明，除了兼容于C的低阶特性外，C++提供的高级特性以及在此基础上发展的各种惯用法可以让我们编写出更加简洁、优雅、高效、健壮的程序。

这些高级特性和惯用法包括精致且高效的标准库和各种“准标准库”，与效率、健壮性、异常安全等主题有关的各种惯用法，以及在C++的未来占据更重要地位的模板和泛型程序设计技术等。它们发展于力量强大的C++社群，并被这个社群中最负声望的专家提炼、升华成一本本精彩的著作。毫无疑问，这些学术成果必将促进C++社群创造出更多的实践成果。

我个人认为，包括操作系统、设备驱动、编译器、系统工具、图像处理、数据库系统以及通用办公软件等在内的基础软件更能够代表一个国家的软件产业发展质量，迄今为止，此类基础性的软件恰好是C++所擅长开发的，因此，可以感性地说，C++的应用水平在一定程度上可以折射出一个国家的软件产业发展水平和健康程度。

前些年国内曾引进出版了一大批优秀的C++书籍，它们拓宽了中国C++程序员的视野，并在很大程度上纠正了长期以来存在于C++的教育、学习和使用方面的种种误解，对C++相关的产业发展起到了一定的促进作用。然而在过去的两年中，随着.NET、Java技术吸引越来越多的注意力，中国软件产业业务化、项目化的状况愈发加剧，擅长于“系统编程”的C++语言的应用领域似有进一步缩减的趋势，这也导致人们对C++的出版教育工作失去了应有的重视。

机械工业出版社华章分社决定继续为中国C++“现代化”教育推波助澜，从2006年起将陆续推出一套“C++设计新思维”丛书。这套丛书秉持精品、高端的理念，其作译者为包括Herb Sutter在内的国内外知名C++技术专家和研究者、教育者，议题紧密围绕现代C++特性，以实用性为主，兼顾实验性和探索性，形式上则是原版影印、中文译著和原创兼收并蓄。每一本书相对独立且交叉引用，篇幅短小却内容深入。作为这套丛书的特邀技术编辑，我衷心希望它们所展示的技术、技巧和理念能够为中国C++社群注入新的活力。

荣耀
2005年12月
南京师范大学
www.royaloo.com

For Tina, my wife and best friend

前　言

场景：布达佩斯。一个炎热的夏天傍晚，穿过多瑙河望去，一直望见东岸。

在本书封面照片所展示的色彩柔和的欧洲风光中，首先映入你眼帘的是哪一栋建筑？几乎可以肯定是左边的国会大厦。这栋巨大的新哥特式建筑以其优美的圆穹、直插云霄的尖塔、大量的外部雕刻以及其他华丽的装饰在刹那之间抓住了你的目光，不过这更可能是因为它与周围的那些多瑙河畔刻板的实用主义建筑形成了鲜明的对比。

为何有此差异？原因首先在于国会大厦竣工于1902年，而其他刻板的实用主义建筑大都是在二战以后至1989年间建成的。

你可能会想，“啊哈，这解释了为何有此差异，不错，但这与《Exceptional C++ Style》到底有啥关系呢？”

毋庸置疑，对风格的表达与注入的哲学和思想倾向有很大的关系，不管我们是在谈论建筑架构还是软件架构，都是如此。我敢肯定你见过有着国会大厦那般规模和华丽的软件，也见过差劲的实用主义软件。极端地说，我还确信你见过不少过于追求风格华而不实的软件，以及很多只顾交差了事的“丑小鸭”（而且永远都不会变成白天鹅）。

风格抑或实用

哪一个更好？

不要过于自信你知道答案。首先，除非你定出明确的标准，否则“更好”只是一个无意义的术语——针对什么而言更好？在哪些情形下更好？其次，答案几乎总是这两者的平衡并以“这取决于……”开始。

本书就是关于如何在C++软件设计和实现的诸多细节方面找到平衡点，以及很好地了解你手头的工具和原料以便知道它们何时适于使用。

请快速回答：与周围单调乏味的建筑相比，国会大厦是更好的建筑吗？它的建筑风格更好吗？在不假思索的情况下你很容易给出肯定的回答——直到你不得不考虑其建造和维修的代价：

- *建造*。在1902年竣工之时，它是世界上最大的国会大厦。人们花费了恐怖的时间、劳动和金钱来建造它，它被很多人认为是“大白象”，意指代价过高的美丽事物。不妨考虑一下：相较而言，花费同样的投资能够建造多少丑陋、单调甚至可能完全令人厌烦的混凝土建筑？别忘了，你工作于这样的一个行业：产品上市的压力远远超过建造该国会大厦所处时代的时间压力。
- *维修*。熟悉国会大厦的人会注意到照片中的建筑正处于翻修中，此项工作已经持续好多年了，极有争议性地耗费了可怕的资金。除了最近这轮代价高昂的翻

修外，此前已有过多次维修。原因在于——悲哀的是，你看到的这座建筑外部美丽的雕刻是用错误的材料制成的：这种材料太软了。在大厦最初竣工后不久，这些雕刻就成了人们持续维修的主题，它们已逐渐被替换为更坚固、更耐久的材料。始于19世纪早期的大规模维修自那以后一直没间断过，持续了近一个世纪。

软件的情形与之类似，重要的是在建造代价与功能之间、在优雅与可维护性之间、在发展的可能性与过分的华丽之间寻找适当的平衡点。

当使用C++实施软件设计和架构时，我们每天都要处理这些以及类似的权衡。在本书探讨的问题中有这样一些：使你的代码异常安全就会使它变得更好吗？果真如此，这里的“更好”是针对什么而言呢？何时它可能并非“更好”？该问题在书中有着明确的答案。封装又如何呢？它使你的软件变得更好吗？为什么？何时封装效果适得其反？如果你对此心存疑惑，请接着读下去。内联是一种好的优化手段吗？它是何时进行的呢？（回答这个问题时你千万慎之又慎。）C++中的export特性和国会大厦有何相通之处？std::string与多瑙河畔的整体建筑之间又有何共同点？

最后，在探讨许多C++技术和特性之后，我们将花费本书末尾一节考察一些摘自公开发布的代码的真实例子，看其作者在哪些方面做得不错，在哪些方面做得欠佳，以及什么样的替代方案可能在实用性和优秀的C++风格之间取得更好的平衡点。

我希望这本书以及《Exceptional C++》系列的其他书籍对你有何帮助呢？呃，我希望它们能够开阔你的视野，扩大你在技术细节及其相互关系方面的知识，并且增加你对如何找到软件开发平衡点的理解。

请再看一眼封面照片的右上方，对，就是那儿！我们都希望乘坐在那样的热气球中，飘荡在城市的上空，饱览美妙的全景。我们将会看看风格和实用性是如何共存、互动、关联和混合的，了解如何进行权衡并找到适当的平衡点，每一个决策都适得其所，并位于一个生机盎然的有机整体中。

是的，我认为布达佩斯是一个伟大的城市，它具有如此丰富的历史，它充满无尽的隐喻。

非凡的苏格拉底

古希腊哲学家苏格拉底通过向学生提问进行教学。那些问题被设计用于引导他们，并帮助他们从已知的东西得出结论，还向他们展示正在学习的东西是如何相互联系的，又如何与他们已有的知识相互联系。这种教学方式是如此著名，以至于今天我们称之为“苏格拉底教学法”。从我们作为学生的观点来看，苏格拉底传奇性的教学方法能够引起我们的兴趣，促使我们思考，帮助我们联系并应用已知的知识去获取新知识。

本书与其姊妹篇《Exceptional C++》[Sutter00]和《More Exceptional C++》

[Sutter02]一样，借鉴了苏格拉底的方法。本书假定你正身处编写产品级C++软件的某些领域，它使用问答的形式教你如何有效地使用标准C++及其标准库，并特别关注于在现代C++中实施健全的软件工程。很多问题都直接来源于我和其他人在编写产品级C++代码时所积累的经验。问题的目标在于帮助你从已知的以及刚学到的东西中得出结论，并展示它们之间是如何相互联系的。而谜题则向你展示如何对C++设计和编程问题进行理性的分析。其中有些是常见的议题，有些则不那么常见；有些问题很简单，有些则比较深奥；还有一些问题只是因为它们有趣才出现于书中。

本书涵盖C++的方方面面。我的意思并不是说它触及了C++的每一个细节（那将需要多得多的篇幅），而是指它从C++语言和库特性的广阔的原料中选取素材，向你展示看似无关的特性是如何被综合使用的，从而形成针对常见问题的新颖的解决方案。本书还展示了那些看似无关的部分是如何相互关联的（即便有时你不希望它们之间有什么关联）以及如何处理这些关联。你将在本书中看到关于模板与名字空间、异常与继承、健壮的类设计与设计模式、泛型编程与宏的使用魔法等内容。它们并非以随意花边新闻的形式出现，而是以具有内在联系的条款的形式，向你展示现代C++中所有这些部分之间的相互关系。

《Exceptional C++ Style》从《Exceptional C++》和《More Exceptional C++》停步的地方继续前行。本书继承了前两本书的传统：它以短小精悍的条款为组织形式，并将这些条款分组为主题明确的章节，从而介绍新知识。读过前两本书的读者会发现一些熟悉的章节和主题，不过现在包含了新内容，例如异常安全、泛型编程以及优化和内存管理技术等。这几本书在结构和主题而非内容上有重叠之处。本书持续了对泛型编程和有效地使用C++标准库的强调，并包括对一些重要的模板和泛型编程技术的讨论。

书中的大多数条款最初出现于杂志专栏和因特网上，尤其出自我为《C/C++ Users Journal》、《Dr. Dobb's Journal》、《C++ Report》（已停刊）以及其他出版物撰写的专栏和文章，以及我的“Guru of the Week”[GotW]的议题63到86。与最初的版本相比，本书中的材料经过了重大的修订、扩充、校正和更新，因而本书（连同www.gotw.ca上不可或缺的勘误表）应被视作那些原始材料的最新权威版。

我假定你已经知道的

我期望你已经掌握了C++基础知识，如果你还没有，可以从一本介绍性和概览性的C++好书开始学习。像Bjarne Stroustrup的《The C++ Programming Language》（第3版）[Stroustrup00]或Stan Lippman和Josée Lajoie合著的《C++ Primer》（第3版）[Lippman98]这样的经典著作都是不错的选择。接下来，务必选读一本编程风格指南，例如Scott Meyers的经典著作《Effective C++》系列[Meyers96, Meyers97]。我发现基于浏览器的CD版[Meyers99]方便且实用。

如何阅读本书

书中的每一个条款都以谜题或问题的形式呈现，并带有一个介绍性的头部，如下所示：

##. 本条款的标题 Difficulty: #

对该条款将要讨论的内容的简短介绍。

标题和难度等级提示你将要遭遇到什么。在主要问题（即Guru问题）之前通常是介绍性或回顾性的问题（即“JG问题”，其中JG（junior-grade）意指新来的下级军官）。注意，难度等级只是我自己预期大多数人认为问题有多难的主观猜测，因此，你也许会发现对你而言一个难度等级为7的问题比某个难度等级为5的问题更简单。实际上，自从写作《Exceptional C++》[Sutter00]和《More Exceptional C++》[Sutter02]以来，我就不断收到一些电子邮件，说“条款N比你说的要容易（或困难）！”对于同一个条款而言，不同的人认为“更容易”或“更困难”是很正常的。难度等级因人而异，任何条款的实际难度取决于你的知识和经验，它们对于其他人而言则可能更容易或更困难一些。话虽如此，大多数情况下你应该发现这些难度等级还是对你将要看到的内容（的难度）给出了相当合理的提示。

你也许打算从头至尾按顺序阅读本书，这很好，但你未必非得这样不可。你也许决定阅读某个特定章节中的所有条款，因为你对该章节的主题特别感兴趣，这同样很好。除了被我称为“小型系列”的那些被标以“Part 1”、“Part 2”等条款讨论的是相关的问题外，书中条款通常是相当独立的，因此你可以遵循条款之间的交叉引用以及对前两本“Exceptional C++”系列书籍的引用，自由地跳着读。我唯一要忠告的是，那些“小型系列”要成组按顺序阅读，除此之外，如何阅读全凭你的喜好。

除非我明确指出某段代码是一个完整的程序，否则它很可能不是。记住，代码示例通常只是代码片段或部分的程序，别指望它们能被独立编译。为了从这些代码片段构建出完整的程序，通常你需要补充一些显而易见的边角代码。

最后，关于URL有必要多说一句：在Web上，东西会动来动去。尤其是，我无法控制的一些内容会动来动去。这使得在印刷书籍上刊印随意的Web URL就变成了真正的痛苦：恐怕在该书下厂印刷之前那些URL就已经过时了，更不要说等它在你的书桌上躺上5年之后了。当我在本书中引用他人的文章或Web站点时，我是通过自己的Web站点（www.gotw.ca）上的URL做到这一点的——我自己的Web站点是我所能控制的，它只包含对实际Web网页的重定向链接。几乎所有在书中引用到的其他作品都已列在参考文献中了，而且在我的网站上还提供了一份具有活动超链接的在线版本。如果你发现印刷在本书中的一个链接不再有效，请写邮件告诉我，我将更新该链接，使其指向新的网页位置（如果我还能找到该网页的话），或者告诉你该网页已不复存在（如

果我找不到的话）。不管怎么说，本书的URL将会保持为最新，尽管在这个因特网世界中印刷媒体的日子是如此难过。呜呼！

致谢

首先感谢我的妻子Tina，感谢她一贯的爱和支持。感谢我的家人在我写作本书和其他作品期间一直陪伴在我左右。即使当我不得不挑灯夜战写作另外一些文章或修改另外一些条款时，他们也总是表现出无尽的耐心。倘若没有他们的耐心和关爱，本书就绝不会有现在这般模样。

我们的小狗Frankie也付出了宝贵的贡献。当我工作时它常常希望我陪它玩，这强迫我不时出来呼吸一些新鲜空气。Frankie对软件架构或程序语言设计甚至代码微观优化一无所知，但它过得非常快乐。

非常感谢丛书编辑Bjarne Stroustrup、编辑Peter Gordon和Debbie Lafferty，还要感谢Tyrrell Albaugh、Bernard Gaffney、Curt Johnson、Chanda Leary-Coutu、Charles Leddy、Malinda McCain、Chuti Prasertsith以及Addison-Wesley团队的其他成员，感谢他们对这个项目的协助和坚持。很难想象还有比他们更好的共事伙伴，他们的热情和协作精神使我对这本书的所有期望都得到了实现。

还有一群人值得感谢和赞扬，就是许多专家审稿人。他们慷慨地提供了富有洞察力的见解并对书稿中的纰漏提出犀利的批评。他们的努力使得你手中的这本书更完整、更可读、更有用。尤其感谢他们向丛书编辑Bjarne Stroustrup提供的技术反馈。还要感谢在本书写作过程中为它的各个部分提出贡献性建议的以下各位人士：Dave Abrahams、Steve Adamczyk、Andrei Alexandrescu、Chuck Allison、Matt Austern、Joerg Barfurth、Pete Becker、Brandon Bray、Steve Dewhurst、Jonathan Caves、Peter Dimov、Javier Estrada, Attila Fehér, Marco Dalla Gasperina, Doug Gregor, Mark Hall, Kevlin Henney, Howard Hinnant、Cay Horstmann、Jim Hyslop、Mark E. Kaminsky、Dennis Mancl、Brian McNamara、Scott Meyers、Jeff Peil、John Potter、P. J. Plauger、Martin Sebor、James Slaughter、Nikolai Smirnov、John Spicer、Jan Christiaan van Winkel、Daveed Vandevoorde以及Bill Wade。书中残存的任何错误、疏忽和不谦虚的双关语，与他们无关，责任全在我自己。

Herb Sutter

2004年5月于西雅图

Preface

The scene: Budapest. A hot summer evening. Looking across the Danube, with a view of the eastern bank.

In the cover photo showing this pastel-colored European scene, what's the first building that jumps out at you? Almost certainly it's the Parliament building on the left. The massive neo-Gothic building catches the eye with its graceful dome, thrusting spires, dozens of exterior statues and other ornate embellishments—and catches the eye all the more so because it stands in stark contrast to the more utilitarian buildings around it on the Danube waterfront.

Why the difference? For one thing, the Parliament building was completed in 1902; the other stark, utilitarian buildings largely date from Hungary's stark and utilitarian Communist era, between World War II and 1989.

"Aha," you might think, "that explains the difference. All very nice, of course, but what on earth does this have to do with *Exceptional C++ Style*?"

Certainly the expression of style has much to do with the philosophy and mindset that goes into it, and that is true whether we're talking about building architecture or software architecture. I feel certain that you have seen software designed on the scale and ornateness of the Parliament building; I feel equally sure that you have seen utilitarian blocky (or should that be "bloc-y"?) software buildings. On the extremes, I am just as convinced that you have seen many gilded lilies that err on the side of style, and many ugly ducklings that err on the side of pushing code out the door (and don't even turn out to be swans).

Style or Substance?

Which is better?

Don't be too sure you know the answer. For one thing, "better" is an unuseful term unless you define specific measures. Better for what? Better in which cases? For another, the answer is almost always a balance of the two and begins with: "It depends..."

This book is about finding that balance in many detailed aspects of software design and implementation in C++, and knowing your tools and materials well to know when they are appropriate.

Quick: Is the Parliament building a *better* building, crafted with *better* style, than the comparatively drab ones around it? It's easy to say yes unthinkingly—until you have to consider building and maintaining it:

- *Construction.* When it was completed in 1902, this was the largest Parliament building in the world. It also cost a horrendous amount of time, effort, and money to produce and was considered by many to be a "white elephant," which means a beautiful thing that comes at too high a cost. Consider: By comparison, how many of the ugly, drab, and perhaps outright boring concrete buildings could have been built for the same investment? And you work in an industry where the time-to-market pressure is far fiercer than the time pressure was in the age of this Parliament.
- *Maintenance.* Those of you familiar with the Parliament will note that in this picture the Parliament building was under renovation and had been in that state for a number of years, at a controversial and arguably ruinous cost. But there's more to the maintenance story than just this recent round of expensive renovations: Sadly, the beautiful sculptures you can see on the exterior of the building were made of the wrong materials, materials that were too soft. Soon after the building was originally completed, those sculptures became the subjects of a continual repair program under which they have been replaced with successively harder and more durable materials, and the heavy maintenance of the "bells and whistles" begun in the early 1900s has gone on continuously ever since—for the past *century*.

Likewise in software, it is important to find the right balance between construction cost and functionality, between elegance and maintainability, between the potential for growth and excessive ornateness.

We deal with these and similar tradeoffs every day as we go about software design and architecture in C++. Among the questions this book tackles are the following: Does making your code exception-safe make it better? If so, for what meanings of "better," and when might it not be better? That question is addressed specifically in this book. What about encapsulation; does it make your software better? Why? When doesn't it? If you're wondering, read on. Is inlining a good optimization, and when is it done? (Be very very careful when you answer this one.) What does C++'s **export** feature have in common with the Parliament building? What does **std::string** have in common with the monolithic architecture of the Danube waterfront in our idyllic little scene?

Finally, after considering many C++ techniques and features, at the end of this book we'll spend our last section looking at real examples of published code and see what the authors did well, what they did poorly, and what alternatives would perhaps have struck a better balance between workmanlike substance and exceptional C++ style.

What do I hope that this and the other *Exceptional C++* books will help to do for you? I hope they will add perspective, add knowledge of details and interrelationships, and add understanding of how balances can be struck in your software's favor.

Please look one more time at the front cover photo, at the top right—that's it, right there. We should want to be in the balloon flying over the city, enjoying the full perspective of the whole view, seeing how style and substance coexist and interact and interrelate and intermingle, knowing how to make the tradeoffs and strike the right balances, each choice in its place in the integral and thriving whole.

Yes, I think Budapest is a great city—so rich with history, so replete with metaphor.

The Exceptional Socrates

The Greek philosopher Socrates taught by asking his students questions—questions designed to guide them and help them draw conclusions from what they already knew and to show them how the things they were learning related to each other and to their existing knowledge. This method has become so famous that we now call it the Socratic method. From our point of view as students, Socrates' legendary approach involves us, makes us think, and helps us relate and apply what we already know to new information.

This book takes a page from Socrates, as did its predecessors, *Exceptional C++* [Sutter00] and *More Exceptional C++* [Sutter02]. It assumes you're involved in some aspect of writing production C++ software today, and it uses a question-answer format to teach how to make effective use of standard C++ and its standard library, with a particular focus on sound software engineering in modern C++. Many of the problems are drawn directly from experiences I and others have encountered while working with production C++ code. The goal of the questions is to help you draw conclusions from things you already know as well as things you've just learned, and to show how they interrelate. The puzzles will show how to reason about C++ design and programming issues—some of them common issues, some not so common; some of them plain issues, some more esoteric; and a couple because, well, just because they're fun.

This book is about all aspects of C++. I don't mean to say that it touches on every detail of C++—that would require many more pages—but rather that it draws from the

wide palette of the C++ language and library features to show how apparently unrelated items can be used together to synthesize novel solutions to common problems. It also shows how apparently unrelated parts of the palette interrelate on their own, even when you don't want them to, and what to do about it. You will find material here about templates and namespaces, exceptions and inheritance, solid class design and design patterns, generic programming and macro magic—and not just as randomized tidbits, but as cohesive Items showing the interrelationships among all these parts of modern C++.

Exceptional C++ Style continues where *Exceptional C++* and *More Exceptional C++* left off. This book follows in the tradition of the first two: It delivers new material, organized in bite-sized Items and grouped into themed sections. Readers of the first book will find some familiar section themes, now including new material, such as exception safety, generic programming, and optimization and memory management techniques. The books overlap in structure and theme but not in content. This book continues the strong emphasis on generic programming and on using the C++ standard library effectively, including coverage of important template and generic programming techniques.

Versions of most Items originally appeared in magazine columns and on the Internet, particularly as print columns and articles I've written for *C/C++ Users Journal*, *Dr. Dobb's Journal*, the former *C++ Report*, and other publications, and also as *Guru of the Week* [GotW] issues #63 to #86. The material in this book has been significantly revised, expanded, corrected, and updated since those initial versions, and this book (along with its de rigueur errata list available at *www.gotw.ca*) should be treated as the current and authoritative version of that original material.

What I Assume You Know

I expect that you already know the basics of C++. If you don't, start with a good C++ introduction and overview. Good choices are a classic tome such as Bjarne Stroustrup's *The C++ Programming Language* [Stroustrup00] or Stan Lippman and Josée Lajoie's *C++ Primer, Third Edition* [Lippman98]. Next, be sure to pick up a style guide such as Scott Meyers' classic *Effective C++* books [Meyers96, Meyers97]. I find the browser-based CD version [Meyers99] convenient and useful.

How to Read This Book

Each Item in this book is presented as a puzzle or problem, with an introductory header that resembles the following:

##. The Topic of This Item

Difficulty: #

A few words about what this Item will cover.

The topic tag and difficulty rating gives you a hint of what you're in for, and typically there are both introductory/review questions ("JG," a term for a new junior-grade military officer) leading to the main questions ("Guru"). Note that the difficulty rating is my subjective guess at how difficult I expect most people will find each problem, so you might well find that a "7" problem is easier for you than some "5" problem. Since writing *Exceptional C++* [Sutter00] and *More Exceptional C++* [Sutter02], I've regularly received e-mail saying that "Item #N is easier (or harder) than that!" It's common for different people to vote "easier!" and "harder!" for the same Item. Ratings are personal; any Item's actual difficulty for you depends on your knowledge and experience and could be easier or harder for someone else. In most cases, though, you should find the rating to be a reasonable guide to what to expect.

You might choose to read the whole book front to back; that's great, but you don't have to. You might decide to read all the Items in a section together because you're particularly interested in that section's topic; that's cool too. Except where there are what I call a "miniseries" of related problems which you'll see designated as Part 1, Part 2, and so on, the Items are pretty independent, and you should feel free to jump around, following the many cross-references among the Items in the book, as well as the many references to the first two *Exceptional C++* books. The only guidance I'll offer is that the miniseries are designed to be read consecutively as a group; other than that, the choice is yours.

Unless I call something a complete program, it's probably not. Remember that the code examples are usually just snippets or partial programs and aren't expected to compile in isolation. You'll usually have to provide some obvious scaffolding to make a complete program out of the snippet shown.

Finally, a word about URLs: On the web, stuff moves. In particular, stuff I have no control over moves. That makes it a real pain to publish random web URLs in a print book lest they become out of date before the book makes it to the printer, never mind after it's been sitting on your desk for five years. When I reference other peoples' articles or web sites in this book, I do it via a URL on my own web site, *www.gotw.ca*, which I can control and which contains just a straight redirect to the real web page. Nearly all the other works I reference are listed in the Bibliography, and I've provided an online version with active hyperlinks. If you find that a link printed in this book no longer works, send me e-mail and tell me; I'll update that redirector to point to the new page's location (if I can find the page again) or to say

that the page no longer exists (if I can't). Either way, this book's URLs will stay up to date despite the rigors of print media in an Internet world. Whew.

Acknowledgments

My thanks go first and most of all to my wife Tina for her enduring love and support, and to all my family for always being there, during this project and otherwise. Even when I had to "go dark" sometimes to crank out another few articles or edit another few Items, their patience knew no bounds. Without their patience and kindness this book would never have come to exist in its current form.

Our little puppy Frankie offered her own valuable contribution, namely wanting to play—even when I was working, thus forcing me to come up for air every once in a while. Frankie knows nothing whatever about software architecture or programming language design or even code micro-optimizations, but she's exuberantly happy anyway. Hmm.

Many thanks to series editor Bjarne Stroustrup, to editors Peter Gordon and Debbie Lafferty, and to Tyrrell Albaugh, Bernard Gaffney, Curt Johnson, Chanda Leary-Coutu, Charles Leddy, Malinda McCain, Chuti Prasertsith, and the rest of the Addison-Wesley team for their assistance and persistence during this project. It's hard to imagine a better bunch of people to work with, and their enthusiasm and cooperation has helped make this book everything I'd hoped it would become.

There is one other group of people who deserve thanks and credit, namely the many expert reviewers who generously offered their insightful comments and savage criticisms on all or part of this material exactly where needed. Their efforts have made the text you hold in your hands that much more complete, more readable, and more useful than it would otherwise have been. Special thanks for their technical feedback to series editor Bjarne Stroustrup, and to the following people who contributed comments on various parts of this material as it was developed: Dave Abrahams, Steve Adamczyk, Andrei Alexandrescu, Chuck Allison, Matt Austern, Joerg Barfurth, Pete Becker, Brandon Bray, Steve Dewhurst, Jonathan Caves, Peter Dimov, Javier Estrada, Attila Fehér, Marco Dalla Gasperina, Doug Gregor, Mark Hall, Kevlin Henney, Howard Hinnant, Cay Horstmann, Jim Hyslop, Mark E. Kaminsky, Dennis Mancl, Brian McNamara, Scott Meyers, Jeff Peil, John Potter, P. J. Plauger, Martin Sebor, James Slaughter, Nikolai Smirnov, John Spicer, Jan Christiaan van Winkel, Daveed Vandevoorde, and Bill Wade. The remaining errors, omissions, and shameless puns are mine, not theirs.

Herb Sutter
Seattle, May 2004

Contents

Preface xi

Generic Programming and the C++ Standard Library 1

1. Uses and Abuses of **vector**
2. The String Formatters of Manor Farm, Part 1: **sprintf** 10
3. The String Formatters of Manor Farm, Part 2: Standard (or Blindingly Elegant) Alternatives 16
4. Standard Library Member Functions 27
5. Flavors of Genericity, Part 1: Covering the Basis [sic] 31
6. Flavors of Genericity, Part 2: Generic Enough? 35
7. Why Not Specialize Function Templates? 42
8. Befriending Templates 49
9. Export Restrictions, Part 1: Fundamentals 59
10. Export Restrictions, Part 2: Interactions, Usability Issues, and Guidelines 68

Exception Safety Issues and Techniques 79

11. Try and Catch Me 80
12. Exception Safety: Is It Worth It? 85
13. A Pragmatic Look at Exception Specifications 89

Class Design, Inheritance, and Polymorphism 99

14. Order, Order! 100
15. Uses and Abuses of Access Rights 104
16. (Mostly) Private 110
17. Encapsulation 119
18. Virtuality 129
19. Enforcing Rules for Derived Classes 139