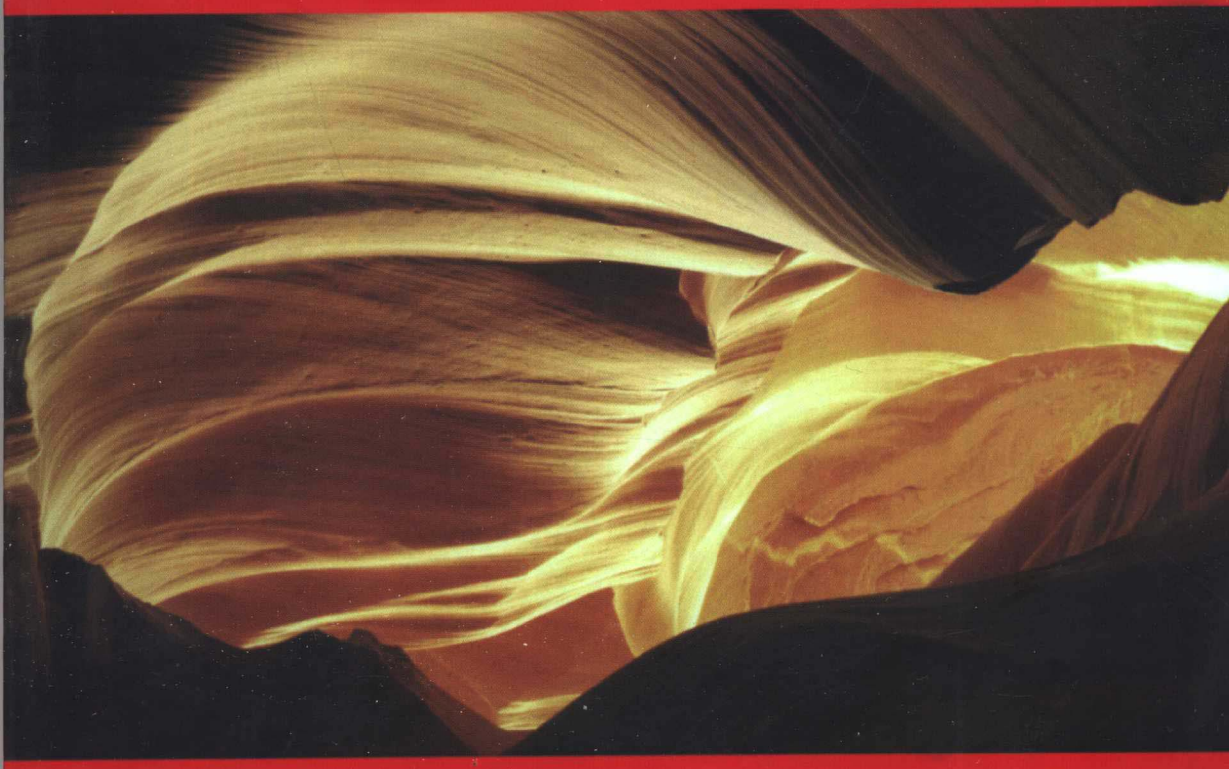
HZ BOOKS
PEARSON Addison Wesley

Modern C++ Design
C++ 设计新思维

Exceptional C++

47 Engineering Puzzles, Programming Problems, and Solutions

（英文版）



（美） Herb Sutter 著

机械工业出版社
China Machine Press

C++
设计新思维

Exceptional C++

(英文版)

47 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：01-2006-0527

图书在版编目（CIP）数据

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

（C++设计新思维）

书名原文：Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions

ISBN 7-111-18369-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004684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 1/16·14.75印张

印数：0 001-3 000册

定价：29.00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

序　言

这是一本非凡的书，不过直至快要读完全书我才意识到它是多么不平凡。这可能是第一本写给已经熟悉C++——熟悉C++的一切——的人看的书。从语言特性到标准库组件再到编程技术，本书从一个主题跳到另一个主题，总是使你处于些微失衡的状态，总是确保你专心致志。就像现实C++程序一样，类设计撞上虚函数的行为，迭代器协定碰到名字查找规则，赋值操作符擦上异常安全，编译依赖遭遇导出模板。就像在现实程序中的一样，语言特性、库组件和编程技术形成的混乱的大漩涡，精彩而令人眩晕。

我将GotW（Guru of the Week）发音为“Gotcha（got you，抓到你了）”可能并无不妥。当我将自己针对书中测验给出的解决方案与Sutter的答案进行比较时，我往往会落入他（和C++）设置的陷阱中，次数之多，使我羞于承认。每当我犯错时，我几乎能看到Herb面带微笑轻声说道：“Gotcha！”也许有人争论这证明我对C++知之甚少，另有人可能宣称这说明C++太复杂，以至于任何人都很难精通它。我则认为这表明在使用C++进行工作时，你必须小心谨慎地思考你正在做的事情。C++是一门威力极大的语言，它被设计用于解决需求苛刻的问题，尽可能细致地磨练你在语言、库和编程惯用法方面的知识，至关重要。本书讨论主题范围之广，内容安排之独特（采用了基于测验的格式），对你的磨练过程将会助一臂之力。

C++新闻组的老读者都知道选出“Guru of the Week”有多么困难，那些有经验的参与者对此更是深有体会。尽管在网上每周只能产生一名guru，然而有了本书提供的知识撑腰，每当编程时你都可冀望产出有着guru质量的代码。

Scott Meyers
1999年6月

前　言

《Exceptional C++》通过例子向你展示如何实施可靠的软件工程。本书包括因特网上流行的C++专题“Guru of the Week”（或简写为GotW）前30个议题的扩充版本，并补充了大量的其他材料。“Guru of the Week”包含一系列独立的C++工程问题和解决方案，它们描述了具体的设计和编程技术。

本书并非盛有代码谜题的摸彩袋，它首先是对现实世界的C++企业软件设计的指南。它使用了问题/解决方案的形式，因为这是我知道的将您——文雅的读者——引入问题背后的思想和指导方针背后的理由最有效的方式。尽管这些条款涵盖了形形色色的主题，然而你会注意到反复出现的主题集中于企业级开发议题上，尤其是异常安全、健全的类和模块设计、适当的优化以及编写遵从标准的可移植代码。

我希望你发现这些内容对你的日常工作有用，我还希望你至少从中发现一些极好的思想和优雅的技术，并且在阅读本书的过程中有时突然喊出“啊哈，原来如此！”是谁说软件工程枯燥乏味的？

如何阅读本书

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好书（像Bjarne Stroustrup的《The C++ Programming Language》（第3版）[㊀]或Stan Lippman和Josée Lajoie合著的《C++ Primer》（第3版）[㊁]这样的经典著作都是不错的选择）开始学习，然后选读一本编程风格指南，例如Scott Meyers的经典著作《Effective C++》（我发现基于浏览器的CD版方便且实用）[㊂]。

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：

Item ##. 谜题的标题　　Difficulty: X

标题和难度等级（通常从3～$9\frac{1}{2}$，最高为10）提示你将要遭遇到什么。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的问题比一个难度等级为5的问题更简单。话虽如此，当你看到一个

㊀ Stroustrup B. *The C++ Programming Language*，第3版（Addison Wesley Longman，1997）。

㊁ Lippman S. and Lajoie J. *C++ Primer*，第3版（Addison Wesley Longman，1998）。

㊂ Meyers S. *Effective C++ CD: 85 Specific Ways to Improve Your Programs and Designs*（Addison Wesley Longman，1999）。可访问http://www.meyerscd.awl.com获得在线演示版。

难度为$9\frac{1}{2}$的怪物浮出水面时，最好还是做最坏的思想准备。

你无需按顺序阅读每一个小节和问题，但有几个地方存在一些相关问题的“小型系列”，你可以看到它们被标以“Part 1”、“Part 2”等，有些地方甚至多至“Part 10”。最好按组阅读这些小型系列。

本书包含很多指导方针，在这些指导方针中，以下单词通常传达特别的含义：

- always：这是绝对必需的。永远恪守之。
- prefer：这通常是正确的方式。只有当某种情形有着特别的正当理由时才使用别的方式。
- consider：这可能有用也可能没用，不过它值得考虑。
- avoid：这通常不是最佳方式，甚至可能有危险。寻求替代方案，只有当某种情形有着明确的保证时才这么做。
- never：这极其糟糕，想都不要想，否则会成为你职业生涯的绊脚石。

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。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！

来龙去脉：GotW与PeerDirect

C++“Guru of the Week”系列已经由来已久。GotW最初创建于1996年底，为我们自己在PeerDirect的开发团队提供有趣的挑战和继续教育。我编写它是为了提供有趣味的学习工具，包括对继承和异常安全之类特性的正确用法的讲解。随着时间的推移，我还将它作为一种工具，用于向我们的团队介绍C++标准会议正在进行的改动。从那以后，GotW作为因特网新闻组comp.lang.c++.moderated的定期专栏对一般C++公众开放，在那儿你可以找到每一个新议题的问题和答案（以及大量有趣的讨论）。

在PeerDirect用好C++非常重要，这与在你的公司用好C++的重要性有很多相同的理由，尽管要达到的目标可能并不相同。我们构建用于分布式数据库和数据库复制的系统软件，在这些领域，诸如可靠性、安全性、可移植性、效率，以及其他很多企业级议题，都是生死攸关的考虑。我们编写的软件要能够移植到不同的编译器和操作系统上，当发生数据库事务死锁、通信中断以及程序异常时，它要保持安全、强健。顾客用它来管理位于智能卡和pop终端或PalmOS和WinCE设备上的微型数据库，或管理部门级Windows

NT和Linux、Solaris服务器，甚至管理Web服务器和数据仓库的大规模并行Oracle后端。这些都使用同样的软件、同样的代码，对可靠性有着同样的要求。现在，当我们费力地工作于大量的密集而未加注释的代码之上时，就会面临可移植性和可靠性的挑战。

对于读者中过去几年来已经读过因特网上“Guru of the Week”的人，我有两句话要说：

- 感谢你们的关心、支持、电子邮件、赞扬、纠正、评论、批评、提问，特别感谢你们对GotW系列印行成书的要求。它来了，希望你们喜欢。
- 本书包含的内容比你在网上看过的多得多。

《Exceptional C++》并不只是对已经浮现于网络空间的陈旧的GotW议题的剪贴。所有问题和解决方案都已在相当大程度上修订和重写，例如，条款8到条款17讨论的异常安全起先出现于单个GotW谜题中，现在则变成了一个具有10个部分的小型深入系列。每一个问题和解决方案都已经被检视，使其跟上修改后的正式C++标准。

所以，如果你以前是GotW的定期读者，本书中仍然有大量的新东西。再一次对所有忠实的读者表示感谢，希望这份材料能够帮助你继续磨炼、扩展你的软件工程和C++编程技能。

致谢

首先当然要感谢comp.lang.c++.moderated 上的所有GotW的读者和热爱者，尤其是那些参加为本书起名竞赛的胜出者。有两位对引导我们定出最终书名帮助尤巨，我要特别对他们表示感谢：Marco Dalla Gasperina建议取名“Enlightened C++”，Rob Stewart则建议取名“Practical C++ Problems and Solutions”。鉴于这里反复强调异常安全，更进一步加入双关语“exceptional”是自然而然的。

非常感谢丛书编辑Bjarne Stroustrup，感谢Marina Lang、Debbie Lafferty以及Addison Wesley Longman的其余编辑人员，感谢他们对此项目投入持续的热情以及在1998年Santa Cruz C++标准会议上给予的周到的招待。

我还要感谢很多审稿人（其中不少人是C++标准委员会的成员），他们提供了深刻而尖锐的评论，对改善你将要看到的正文内容很有帮助。特别感谢Bjarne Stroustrup和Scott Meyers，以及Andrei Alexandrescu、Steve Clamage、Steve Dewhurst、Cay Horstmann、Jim Hyslop、Brendan Kehoe、Dennis Mancl，感谢他们宝贵的洞察力和评论。

最后，特别感谢我的家人和朋友，感谢你们一直以各种方式陪伴着我。

Herb Sutter

1999年6月

Foreword

This is a remarkable book, but it wasn't until I had nearly finished reading it that I realized just how remarkable it is. This could well be the first book ever written for people who are already familiar with C++—*all* of C++. From language features to components of the standard library to programming techniques, this book skips from topic to topic, always keeping you slightly off balance, always making sure you're paying attention. Just like real C++ programs. Class design bumps into the behavior of virtual functions, iterator conventions run up against name lookup rules, assignment operators sideswipe exception safety, compilation dependencies cross paths with exported templates. Just like they do in real programs. The result is a dizzying maelstrom of language features, library components, and programming techniques at once both chaotic and magnificent. Just like real programs.

I pronounce *GotW* such that it rhymes with "Gotcha," and perhaps that's fitting. As I compared my solutions to the book's quizzes against Sutter's answers, I fell into the traps he (and C++) laid before me more often than I'd like to admit. I could almost see Herb smiling and softly saying "Gotcha!" for each error I made. Some may argue that this proves I don't know much about C++. Others may claim it demonstrates that C++ is too complex for anyone to master. I believe it shows that when you're working in C++, you have to think carefully about what you're doing. C++ is a powerful language designed to help solve demanding problems, and it's important that you hone your knowledge of the language, its library, and its programming idioms as finely as you can. The breadth of topics in this book will help you do that. So will its unique quiz-based format.

Veteran readers of the C++ newsgroups know how difficult it is to be proclaimed a *Guru of the Week*. Veteran participants know it even better. On the Internet, of course, there can be only one guru each week, but, backed by the information in this book, you can reasonably hope to produce guru-quality code every time you program.

Scott Meyers
June 1999

Preface

Exceptional C++ shows by example how to go about solid software engineering. Along with a lot of other material, this book includes expanded versions of the first 30 issues of the popular Internet C++ feature *Guru of the Week* (or, in its short form, *GotW*), a series of self-contained C++ engineering problems and solutions that illustrate specific design and coding techniques.

This book isn't a random grab-bag of code puzzles; it's primarily a guide to sound real-world enterprise software design in C++. It uses a problem/solution format because that's the most effective way I know to involve you, gentle reader, in the ideas behind the problems and the reasons behind the guidelines. Although the Items cover a variety of topics, you'll notice recurring themes that focus on enterprise development issues, particularly exception safety, sound class and module design, appropriate optimization, and writing portable standards-conforming code.

I hope you find this material useful in your daily work. But I also hope you find at least a few nifty thoughts and elegant techniques, and that from time to time, as you're reading through these pages, you'll suddenly have an "Aha! Gnarly!" moment. After all, who says software engineering has to be dull?

How to Read This Book

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 like Bjarne Stroustrup's *The C++ Programming Language, Third Edition*[1] or Stan Lippman and Josée Lajoie's *C++ Primer, Third Edition*[2]), and then be sure to pick up a style guide like Scott Meyers' classic *Effective C++* books (I find the browser-based CD version convenient and useful).[3]

---

1. Stroustrup B. *The C++ Programming Language, Third Edition* (Addison Wesley Longman, 1997).
2. Lippman S. and Lajoie J. *C++ Primer, Third Edition* (Addison Wesley Longman, 1998).
3. Meyers S. *Effective C++ CD: 85 Specific Ways to Improve Your Programs and Designs* (Addison Wesley Longman, 1999). An online demo is available at http://www.meyerscd.awl.com.

Each item in this book is presented as a puzzle or problem, with an introductory header that looks like this:

ITEM ##: THE TOPIC OF THIS PUZZLE **DIFFICULTY: X**

The topic tag and difficulty rating (typically anything from 3 to 9½, based on a scale of 10) gives you a hint of what you're in for. Note that the difficulty rating is my own subjective guess at how difficult I expect most people will find each problem, so you may well find that a given 7 problem is easier for you than another 5 problem. Still, it's better to be prepared for the worst when you see a 9½ monster coming down the pike.

You don't have to read the sections and problems in order, but in several places there are "miniseries" of related problems that you'll see designated as "Part 1," "Part 2," and so on—some all the way up to "Part 10." Those miniseries are best read as a group.

This book includes many guidelines, in which the following words usually carry a specific meaning:

- **always** = This is absolutely necessary. Never fail to do this.
- **prefer** = This is usually the right way. Do it another way only when a situation specifically warrants it.
- **consider** = This may or may not apply, but it's something to think about.
- **avoid** = This is usually not the best way, and might even be dangerous. Look for alternatives, and do it this way only when a situation specifically warrants it.
- **never** = This is extremely bad. Don't even think about it. Career limiting move.

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's, never mind after it's been sitting on your desk for five years. When I reference other people's 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. 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.

---

3. Meyers S. *Effective C++ CD: 85 Specific Ways to Improve Your Programs and Designs* (Addison Wesley Longman, 1999). An online demo is available at http://www.meyerscd.awl.com.

How We Got Here: *GotW* and PeerDirect

The C++ *Guru of the Week* series has come a long way. *GotW* was originally created late in 1996 to provide interesting challenges and ongoing education for our own development team here at PeerDirect. I wrote it to provide an entertaining learning tool, including rants on things like the proper use of inheritance and exception safety. As time went on, I also used it as a means to provide our team with visibility to the changes being made at the C++ standards meetings. Since then, *GotW* has been made available to the general C++ public as a regular feature of the Internet newsgroup *comp.lang.c++.moderated*, where you can find each new issue's questions and answers (and a lot of interesting discussion).

Using C++ well is important at PeerDirect for many of the same reasons it's important in your company, if perhaps to achieve different goals. We happen to build systems software—for distributed databases and database replication—in which enterprise issues such as reliability, safety, portability, efficiency, and many others are make-or-break concerns. The software we write needs to be able to be ported across various compilers and operating systems; it needs to be safe and robust in the presence of database transaction deadlocks and communications interruptions and programming exceptions; and it's used by customers to manage tiny databases sitting inside smart cards and pop machines or on PalmOS and WinCE devices, through to departmental Windows NT and Linux and Solaris servers, through to massively parallel Oracle back-ends for Web servers and data warehouses—with the same software, the same reliability, the same code. Now *that's* a portability and reliability challenge, as we creep up on half a million tight, noncomment lines of code.

To those of you who have been reading *Guru of the Week* on the Internet for the past few years, I have a couple of things to say:

- Thank you for your interest, support, e-mails, kudos, corrections, comments, criticisms, questions—and especially for your requests for the *GotW* series to be assembled in book form. Here it is; I hope you enjoy it.
- This book contains *a lot* more than you ever saw on the Internet.

Exceptional C++ is not just a cut-and-paste of stale *GotW* issues that are already floating out there somewhere in cyberspace. All the problems and solutions have been considerably revised and reworked—for example, Items 8 through 17 on exception safety originally appeared as a single *GotW* puzzle and have now become an in-depth, 10-part miniseries. Each problem and solution has been examined to bring it up to date with the then-changing, and now official, C++ standard.

So, if you've been a regular reader of *GotW* before, there's a lot that's new here for you. To all faithful readers, thanks again, and I hope this material will help you continue to hone and expand your software engineering and C++ programming skills.

Acknowledgments

First, of course, thanks to all the *GotW* readers and enthusiasts on *comp.lang.c++.moderated*, especially the scores of people who participated in the contest to select a name for this book. Two in particular were instrumental in leading us to the final title, and I want to

thank them specifically: Marco Dalla Gasperina for suggesting the name *Enlightened C++*, and Rob Stewart for suggesting the name *Practical C++ Problems and Solutions*. It was only natural to take these a step further and insert the pun *exceptional*, given the repeated emphasis herein on exception safety.

Many thanks also to series editor Bjarne Stroustrup and to Marina Lang, Debbie Lafferty, and the rest of the Addison Wesley Longman editorial staff for their continued interest and enthusiasm in this project, and for hosting a really nice reception at the Santa Cruz C++ standards meeting in 1998.

I also want to thank the many people who acted as reviewers—many of them fellow standards-committee members—who provided thoughtful and incisive comments that have helped to improve the text you are about to read. Special thanks to Bjarne Stroustrup and Scott Meyers, and to Andrei Alexandrescu, Steve Clamage, Steve Dewhurst, Cay Horstmann, Jim Hyslop, Brendan Kehoe, and Dennis Mancl, for their invaluable insights and reviews.

Finally, thanks most of all to my family and friends for always being there, in so many different ways.

Herb Sutter
June 1999

Contents

Generic Programming and the C++ Standard Library

To begin, let's consider a few selected topics in the area of generic programming. These puzzles focus on the effective use of templates, iterators, and algorithms, and how to use and extend standard library facilities. These ideas then lead nicely into the following section, which analyzes exception safety in the context of writing exception-safe templates.

Item 1: Iterators — Difficulty: 7

Every programmer who uses the standard library has to be aware of these common and not-so-common iterator mistakes. How many of them can you find?

The following program has at least four iterator-related problems. How many can you find?

```
int main()
{
  vector<Date> e;
  copy( istream_iterator<Date>( cin ),
        istream_iterator<Date>(),
        back_inserter( e ) );
  vector<Date>::iterator first =
        find( e.begin(), e.end(), "01/01/95" );
  vector<Date>::iterator last =
        find( e.begin(), e.end(), "12/31/95" );
  *last = "12/30/95";
  copy( first,
        last,
        ostream_iterator<Date>( cout, "\n" ) );
  e.insert( --e.end(), TodaysDate() );
  copy( first,
        last,
        ostream_iterator<Date>( cout, "\n" ) );
}
```

Solution

```
int main()
{
  vector<Date> e;
  copy( istream_iterator<Date>( cin ),
        istream_iterator<Date>(),
        back_inserter( e ) );
```

This is fine so far. The `Date` class writer provided an extractor function with the signature `operator>>( istream&, Date& )`, which is what `istream_iterator<Date>` uses to read the `Date`s from the `cin` stream. The `copy()` algorithm just stuffs the `Date`s into the vector.

```
  vector<Date>::iterator first =
        find( e.begin(), e.end(), "01/01/95" );
  vector<Date>::iterator last =
        find( e.begin(), e.end(), "12/31/95" );
  *last = "12/30/95";
```

Error: This may be illegal, because `last` may be `e.end()` and therefore not a dereferenceable iterator.

The `find()` algorithm returns its second argument (the end iterator of the range) if the value is not found. In this case, if "12/31/95" is not in `e`, then `last` is equal to `e.end()`, which points to one-past-the-end of the container and is not a valid iterator.

```
  copy( first,
        last,
        ostream_iterator<Date>( cout, "\n" ) );
```

Error: This may be illegal because `[first,last)` may not be a valid range; indeed, `first` may actually be after `last`.

For example, if "01/01/95" is not found in `e` but "12/31/95" is, then the iterator `last` will point to something earlier in the collection (the `Date` object equal to "12/31/95") than does the iterator `first` (one past the end). However, `copy()` requires that `first` must point to an earlier place in the same collection as `last`—that is, `[first,last)` must be a valid range.

Unless you're using a checked version of the standard library that can detect some of these problems for you, the likely symptom if this happens will be a difficult-to-diagnose core dump during or sometime after the `copy()`.

```
  e.insert( --e.end(), TodaysDate() );
```

First error: The expression "`--e.end()`" is likely to be illegal.

The reason is simple, if a little obscure: On popular implementations of the standard library, `vector<Date>::iterator` is often simply a `Date*`, and the C++ language doesn't