HZ BOOKS

PEARSON Addison Wesley

Modern C++ Design C++ 设计新思维

Accelerated C++

Practical Programming by Example

（英文版）

（美）Andrew Koenig
Barbara E. Moo 著

机械工业出版社
China Machine Press

Modern C++ Design
C++
设计新思维

Accelerated C++

(英文版)

Practical Programming by Example

(美) Andrew Koenig
Barbara E. Moo 著

机械工业出版社
China Machine Press

English reprint edition copyright © 2006 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Accelerated C++: Practical Programming by Example* (ISBN 0-201-70353-X) by Andrew Koenig and Barbara E. Moo, Copyright © 2000.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison-Wesley.

For sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macau SAR).

本书英文影印版由Pearson Education Asia Ltd. 授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

本书封面贴有Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

版权所有，侵权必究。

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2006-1935

图书在版编目（CIP）数据

Accelerated C++（英文版）/（美）凯尼格（Koenig, A.）等著. -北京：机械工业出版社，2006.4

（C++设计新思维）

书名原文：Accelerated C++: Practical Programming by Example

ISBN 7-111-18831-4

Ⅰ. A…　Ⅱ. 凯…　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029825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

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2006年 4 月第 1 版第 1 次印刷

718mm × 1020mm 1/16 · 22印张

定价：42.00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

前　言

一种教授C++编程的新方式

我们假定你希望快速地学习如何编写有用的C++程序，因此我们从解释C++最有用的部分开始。尽管一旦付诸实施，这种策略看上去便理所当然，但它其实有着激进的隐喻：我们并不从C开始教C++，纵然C++构建于C之上。相反，我们从一开始就使用高级数据结构，只是在较晚的时候才解释那些数据结构所依赖的基础。这种方式可以让你很快就编写出地道的C++程序。

从另一方面来看我们的方式也非同寻常：我们专注于解决问题而非探究语言和库的特性。当然，我们也解释一些特性，但这么做是为了给程序提供支持，而不是将程序用作演示语言特性的工具。

正因为本书教授C++编程而非仅仅讲解语言特性，所以对那些已经了解一些C++并希望以更自然、更有效的风格使用C++的读者而言，这本书尤其有用。C++新手往往只学习语言技巧而没有学习如何将其用于解决日常问题。

我们的方式同时适用于初学者和经验丰富的程序员

在过去的每一个夏季我们都在斯坦福大学教授为期一周的C++强化班。起先我们采用传统的方式进行授课，即假定学生已经了解C语言，因此从讲授如何定义类开始，然后系统地过渡到语言的其余部分。我们发现，在所学的知识足以编写出有意义的程序之前，学生们大约要熬过困惑、沮丧的两三天。而一旦能够写出有意义的程序，他们学起来就快多了。

当我们接触到一种为崭新的标准库提供足够支持的C++实现时，我们就全面修订了课程。新课程从一开始就使用标准库，专注于编写有用的程序，仅当学生掌握的知识足以富有成效地运用语言和库的各种细节时，我们才深入探讨有关细节。

结果富有戏剧性：仅在教室端坐一天，学生们就能编写出在老课程中要花费大半周时间才能写出来的程序。此外，他们的沮丧感也烟消云散、了无踪影。

抽象

我们的方式之所以可行，是因为C++（以及我们对它的理解）已经日渐成熟，这使我们得以忽略许多为早期的C++程序和程序员所依赖的低层概念。

允许忽略细节的能力是成熟技术的特征。例如，早期的汽车常常会出故障，因此每一个司机都不得不同时是一个业余的机械修理工。在不知道车子出了故障如何修好它并

试读结束：需要全本请在线购买：　www.ertongbook.com

将其开回家的情况下，驾车出行实为莽撞之举。但今天的司机无需具备工程细节知识即可从事汽车运输。也许他们出于某些原因去学习工程细节，但那完全是另外一回事。

我们将“抽象”定义为有选择性的忽略，即专注于和手头任务相关的概念并忽略其他一切，我们认为这是现代程序设计中最重要的思想。编写一个成功的程序的关键在于明白问题的哪些部分应该予以考虑，哪些部分又应该被忽略。每一种编程语言都提供了用于创建有用的抽象的工具，每一个成功的程序员都清楚如何使用这些工具。

我们认为抽象非常有用，于是书中随处可见“抽象”。当然，我们通常并不直接称之为抽象，它们以多种形式出现，分别称为函数、数据结构、类以及继承——所有这些都是抽象。我们不仅提到它们，而且对它们的使用贯穿全书。

如果抽象有着良好的设计并且是善加选择的结果，我们相信即使在不了解其工作机理的全部细节的情况下也可以使用它们。我们无需成为汽车工程师就可以驾驶汽车，类似地，我们在能够使用C++之前也无需了解其全部工作细节。

涵盖范围

如果你将C++编程视作一件严肃的事情，就需要知道本书包含的一切知识，尽管本书并没有告诉你需要知道的一切知识。

这种说法并不像听上去那样矛盾。没有哪一本具有这般厚度的书能够涵盖你需要知道的一切C++知识。由于不同的程序员和应用需要不同的知识，因而任何一本覆盖了C++所有知识的书籍（例如Stroustrup的《The C++ Programming Language》，Addison-Wesley，2000）都不可避免地告诉你许多你本不需要知道的东西。原因是明摆着的，即便你不需要那些知识，自有他人需要。

另一方面，C++的许多部分具有普遍的重要意义，如果不理解它们就很难高效地开发程序。我们专注于这些部分。仅仅使用本书提供的信息来编写各种各样的有用程序是完全可能的。事实上，本书的一位审稿人（一个使用C++编写真实商业系统的主程序员）告诉我们，本书基本上涵盖了他在工作中用到的所有编程设施。

使用这些设施，你可以编写真正的C++程序而非C或任何其他语言风格的程序。一旦你掌握了书中介绍的内容，就会明白自己还需要学一些什么别的C++知识以及如何去学。在业余望远镜制作者中流传着这么一种说法：先制作一个3英寸的镜片然后再制作6英寸的镜片比一上来就制作一个6英寸的镜片要来得容易。C++的学习与之类似。

我们仅仅讨论标准C++而忽略其他专有扩展。这种方式的优点在于我们教你编写的程序可以在任何环境下工作。不过这也意味着我们不会探讨如何编写运行在窗口环境下的程序，因为这样的程序将不可避免地被绑定到某种特定的环境，而且往往还被绑定到某个特定的厂商。如果你希望编写仅工作于特定环境下的程序，那么需要改用其他途径学习有关编程方法——然而请不要就此合上本书，因为我们在书中介绍的方式具有普遍性，有朝一日你可以在任何环境中使用在此学到的任何知识。你当然可以转而阅读其他关于GUI应用的书，不过在此之前请首先阅读这一本。

富有经验的C和C++程序员请注意

当学习一门新的编程语言时，你可能抗拒不了采用类似于你已经知道的其他语言的风格来编写程序的诱惑。我们的方式通过从一开始就使用来自C++标准库的高级抽象来努力避免这种“诱惑”的出现。如果你已经是一个富有经验的C或C++程序员，这种方式包含一些好消息和一些坏消息，它们其实是相同的消息。

消息就是，你可能会对这一点感到惊讶：你已有的C++知识对于理解我们展示的C++知识用处不大。一开始你需要学习的知识要比你预想的多（此为坏消息），但你的学习效率也将比你预期的要高（此为好消息）。特别是，如果你此前学过C++，你可能首先学习如何采用C进行编程，这就意味着你的C++编程风格是建立在C的基础之上的。这种方式并没有错，但我们的方式是如此与众不同，相信你将看到你以前从未见过的C++的另一面。

当然了，很多语法细节是相似的，不过它们仅仅是细节而已。我们处理重要思想的顺序全然不同于你此前遇到过的。举个例子，我们直至第10章才提到指针或数组，我们甚至压根就没打算讨论你的旧爱printf和malloc。另一方面，我们在第1章就开始讨论标准库的string类。总之，当我们宣称正在采用一种全新的教授方式时，事实的确如此！

本书的结构

你也许会发现将本书分成两个部分进行考虑会更方便一些。第一部分为前七章，专注于使用标准库抽象进行编程。第二部分从第8章开始，讨论如何定义你自己的抽象。

首先介绍库是一个非同寻常的主意，但我们认为这是正确的。C++语言的许多部分——尤其是那些较困难的部分——主要出于库作者利益的考虑而存在的。库的用户根本无需了解语言的那些部分。因此在第一部分中我们避开语言的这些特性不谈，与使用较传统的方式相比，我们的方式使得更快地编写出有意义的C++程序成为可能。

一旦理解了库的用法，你就要准备学习库赖以构建的低层设施以及如何使用这些设施来编写你自己的库。此外，你将会获得对如何使得一个库有用以及何时应避免从头编写全新的库的感性认识。

尽管本书要比许多C++书籍薄，但我们在书中已努力使用每一个重要的概念至少两次，关键概念的使用次数则更多。结果，书中的许多部分会引用到其他部分。这种引用看上去像“§39.4.3/857”的模样，它指的是引用的文本位于第857页，并且是第39.4.3小节的一部分（如果本书有这么多的小节或页数的话，含义就是如此）。当第一次解释某个概念时，我们会采用粗斜体字标明，使得它易被发现并可引起你的注意，以将其视作一个重点。

本书的每一章（除了最后一章外）都以一个称为“Details”的小节收尾。安排这一节有两个用意：它们可以使你加深对该章介绍的概念的记忆，并且它们还包含一些额外的相关信息，我们认为你终有一天需要了解这些信息。建议初次阅读时忽略这些内容，他日需要时可以再回首查阅。

本书的两个附录在细节的层面上总结并阐明了语言和库的重要部分，希望在你编写程序时它们能派上用场。

最大限度地发挥本书的作用

每一本关于编程的书都包含有示例程序，本书也不例外。为了理解这些程序是如何工作的，除了在计算机上运行它们外没有更好的方式。这样的计算机到处都有，而且新式计算机也不断出现。这就意味着，等你读到这些文字时，我们对它们的任何叙述都可能是不准确的。如果你还不知道如何编译和执行一个C++程序，请访问http://www.acceleratedcpp.com并参考那里的描述。我们会不时地更新网站，为之添加关于运行C++程序机制的建议和信息。该网站还提供了一些机器可读的示例程序版本以及其他一些你也许感觉有趣的信息。

致谢

我们对以下人士表示谢意，没有他们本书就不可能诞生。本书的成型很大程度上要归功于以下审稿人：Robert Berger、Dag Brück、Adam Buchsbaum、Stephen Clamage、Jon Kalb、Jeffrey Oldham、David Slayton、Bjarne Stroustrup、Albert Tenbusch、Bruce Tetelman以及Clovis Tondo。Addison-Wesley有许多工作人员都参与了本书的出版工作，我们知道的有Tyrrell Albaugh、Bunny Ames、Mike Hendrickson、Deborah Lafferty、Cathy Ohala以及Simone Payment等。Alexander Tsiris则校核了13.2.2节中的希腊词源。最后，以高级编程开始讲解的想法由来已久，这是受到数百名耐心听完我们课程的学生以及数千名听过我们演讲的人们的激励而产生的。

Andrew Koenig
Barbara E. Moo
于新泽西州吉列
2000年6月

Preface

A new approach to C++ programming

We assume that you want to learn quickly how to write useful C++ programs. Therefore, we start by explaining the most useful parts of C++. This strategy may seem obvious when we put it that way, but it has the radical implication that we do not begin by teaching C, even though C++ builds on C. Instead, we use high-level data structures from the start, explaining only later the foundations on which those data structures rest. This approach lets you to begin writing idiomatic C++ programs immediately.

Our approach is unusual in another way: We concentrate on solving problems, rather than on exploring language and library features. We explain the features, of course, but we do so in order to support the programs, rather than using the programs as an excuse to demonstrate the features.

Because this book teaches C++ programming, not just features, it is particularly useful for readers who already know some C++, and who want to use the language in a more natural, effective style. Too often, people new to C++ learn the language mechanics without learning how to apply the language to everyday problems.

Our approach works—for beginners and experienced programmers

We used to teach a week-long intensive C++ course every summer at Stanford University. We originally adopted a traditional approach to that course: Assuming that the students already knew C, we started by showing them how to define classes, and then moved systematically through the rest of the language. We found that our students would be confused and frustrated for about two days—until they had learned enough that they could start writing useful programs. Once they got to that point, they learned quickly.

When we got our hands on a C++ implementation that supported enough of what was then the brand-new standard library, we overhauled the course. The new course used the library right from the beginning, concentrated on writing useful programs, and went into details only after the students had learned enough to use those details productively.

The results were dramatic: After one day in the classroom, our students were able to write programs that had taken them most of the week in the old course. Moreover, their frustration vanished.

Abstraction

Our approach is possible only because C++, and our understanding of it, has had time to mature. That maturity has let us ignore many of the low-level ideas that were the mainstay of earlier C++ programs and programmers.

The ability to ignore details is characteristic of maturing technologies. For example, early automobiles broke down so often that every driver had to be an amateur mechanic. It would have been foolhardy to go for a drive without knowing how to get back home even if something went wrong. Today's drivers don't need detailed engineering knowledge in order to use a car for transportation. They may wish to learn the engineering details for other reasons, but that's another story entirely.

We define abstraction as selective ignorance—concentrating on the ideas that are relevant to the task at hand, and ignoring everything else—and we think that it is the most important idea in modern programming. The key to writing a successful program is knowing which parts of the problem to take into account, and which parts to ignore. Every programming language offers tools for creating useful abstractions, and every successful programmer knows how to use those tools.

We think abstractions are so useful that we've filled this book with them. Of course, we don't usually call them abstractions directly, because they come in so many forms. Instead, we refer to functions, data structures, classes, and inheritance—all of which are abstractions. Not only do we refer to them, but we use them throughout the book.

If abstractions are well designed and well chosen, we believe that we can use them even if we don't understand all the details of how they work. We do not need to be automotive engineers to drive a car, nor do we need to understand everything about how C++ works before we can use it.

Coverage

If you are serious about C++ programming, you need to know everything in this book—even though this book doesn't tell you everything you need to know.

This statement is not as paradoxical as it sounds. No book this size can contain everything you'll ever need to know about C++, because different programmers and applications require different knowledge. Therefore, any book that covers all of C++—such as Stroustrup's *The C++ Programming Language* (Addison-Wesley, 2000)—will inevitably tell you a lot that you don't need to know. Someone else will need it, even if you don't.

On the other hand, many parts of C++ are so universally important that it is hard to be productive without understanding them. We have concentrated on those parts. It is possible to write a wide variety of useful programs using only the information in this book. Indeed, one of our reviewers, who is the lead programmer for a substantial commercial system written in C++, told us that this book covers essentially all of the facilities that he uses in his work.

Using these facilities, you can write true C++ programs—not C++ programs in the style of C, or any other language. Once you have mastered the material in this book, you will know enough to figure out what else you want to learn, and how to go about it. Amateur

telescope makers have a saying that it is easier to make a 3-inch mirror and then to make a 6-inch mirror than to make a 6-inch mirror from scratch.

We cover only standard C++, and ignore proprietary extensions. This approach has the advantage that the programs that we teach you to write will work just about anywhere. However, it also implies that we do not talk about how to write programs that run in windowing environments, because such programs are invariably tied to a specific environment, and often to a specific vendor. If you want to write programs that will work only in a particular environment, you will have to turn elsewhere to learn how to do so—but don't put this book down quite yet! Because our approach is universal, you will be able to use everything that you learn here in whatever environments you use in the future. By all means, go ahead and read that book about GUI applications that you were considering—but please read this one first.

A note to experienced C and C++ programmers

When you learn a new programming language, you may be tempted to write programs in a style that is familiar from the languages that you already know. Our approach seeks to avoid that temptation by using high-level abstractions from the C++ standard library right from the start. If you are already an experienced C or C++ programmer, this approach contains some good news and some bad news—and it's the same news.

The news is that you are likely to be surprised at how little of your knowledge will help you understand C++ as we present it. You will have more to learn at first than you might expect (which is bad), but you will learn more quickly than you might expect (which is good). In particular, if you already know C++, you probably learned first how to program in C, which means that your C++ programming style is built on a C foundation. There is nothing wrong with that approach, but our approach is so different that we think you'll see a side of C++ that you haven't seen before.

Of course, many of the syntactic details will be familiar, but they're just details. We treat the important ideas in a completely different order from what you've probably encountered. For example, we don't mention pointers or arrays until Chapter 10, and we're not even going to discuss your old favorites, `printf` and `malloc`, at all. On the other hand, we start talking about the standard-library `string` class in Chapter 1. When we say we're adopting a new approach, we mean it!

Structure of this book

You may find it convenient to think of this book as being in two parts. The first part, through Chapter 7, concentrates on programs that use standard-library abstractions. The second part, starting with Chapter 8, talks about defining your own abstractions.

Presenting the library first is an unusual idea, but we think it's right. Much of the C++ language—especially the harder parts—exists mostly for the benefit of library authors. Library users don't need to know those parts of the language at all. By ignoring those parts of the language until the second part of the book, we make it possible to write useful C++ programs much more quickly than if we had adopted a more conventional approach.

试读结束：需要全本请在线购买： www.ertongbook.com

Once you have understood how to use the library, you will be ready to learn about the low-level facilities on which the library is built, and how to use those facilities to write your own libraries. Moreover, you will have a feeling for how to make a library useful, and when to avoid writing new library code altogether.

Although this book is smaller than many C++ books, we have tried to use every important idea at least twice, and key ideas more than that. As a result, many parts of the book refer to other parts. These references look like §39.4.3/857, which refers to text on page 857 that is part of section 39.4.3—or at least it would do so if this book had that many sections or pages. The first time we explain each idea, we mention it in ***bold italic*** type to make it easy to find and to call your attention to it as an important point.

Every chapter (except the last) concludes with a section called *Details*. These sections serve two purposes: They make it easy to remember the ideas that the chapter introduced, and they cover additional, related material that we think you will need to know eventually. We suggest that you skim these sections on first reading, and refer back to them later as needed.

The two appendices summarize and elucidate the important parts of the language and library at a level of detail that we hope will be useful when you are writing programs.

Getting the most out of this book

Every book about programming includes example programs, and this one is no different. In order to understand how these programs work, there is no substitute for running them on a computer. Such computers abound, and new ones appear constantly—which means that anything we might say about them would be inaccurate by the time you read these words. Therefore, if you do not yet know how to compile and execute a C++ program, please visit `http://www.acceleratedcpp.com` and see what we have to say there. We will update that website from time to time with information and advice about the mechanics of running C++ programs. The site also offers machine-readable versions of some of the example programs, and other information that you might find interesting.

Acknowledgments

We would like to thank the people without whom this book would have been impossible. It owes much of its form to our reviewers: Robert Berger, Dag Brück, Adam Buchsbaum, Stephen Clamage, Jon Kalb, Jeffrey Oldham, David Slayton, Bjarne Stroustrup, Albert Tenbusch, Bruce Tetelman, and Clovis Tondo. Many people from Addison-Wesley participated in its publication; the ones we know about are Tyrrell Albaugh, Bunny Ames, Mike Hendrickson, Deborah Lafferty, Cathy Ohala, and Simone Payment. Alexander Tsiris checked the Greek etymology in §13.2.2/236. Finally, the idea of starting with high-level programs grew over many years, stimulated by the hundreds of students who have sat through our courses and the thousands of people who have attended our talks.

Andrew Koenig
Barbara E. Moo

Gillette, New Jersey
June 2000

Contents

0

Getting started

Let us begin by looking at a small C++ program:

```
// a small C++ program
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Programmers often refer to such a program as a `Hello, world!` program. Despite its small size, you should take the time to compile and run this program on your computer before reading further. The program should write

```
Hello, world!
```

on the standard output, which will typically be a window on your display screen. If you have trouble, find someone who already knows C++ and ask for help, or consult our website, `http://www.acceleratedcpp.com`, for advice.

This program is useful because it is so simple that if you have trouble, the most likely reasons are obvious typographical errors or misconceptions about how to use the implementation. Moreover, thoroughly understanding even such a small program can teach a surprising amount about the fundamentals of C++. In order to gain this understanding, we'll look in detail at each line of the program.

0.1 Comments

The first line of our program is

```
// a small C++ program
```

The `//` characters begin a ***comment***, which extends to the end of the line. The compiler ignores comments; their purpose is to explain the program to a human reader. In this book, we shall put the text of each comment in *italic* type, to make it easier for you to distinguish comments from other parts of the program.

试读结束：需要全本请在线购买： www.ertongbook.com