

C++ 程序设计原理与实践

(英文版)

BJARNE STROUSTRUP

Programming

Principles, and Practice Using C++

(美) Bjarne Stroustrup 著



机械工业出版社
China Machine Press

C++之父Bjarne Stroustrup的最新力作

•为编写实际的应用程序做好准备

无论你是为了进行软件开发还是进行其他领域的工作，本书假定你的最终目标是学会编写实际有用的程序。

•以基本概念和基本技术为重点

与传统的C++教材相比，本书对基本概念和基本技术的介绍更为深入，这会为你编写有用、正确、易维护和有效的代码打下坚实的基础。

•用现代C++语言编程

本书一方面介绍了通用的程序设计方法（包括面向对象程序设计和泛型程序设计），另一方面还对软件开发实践中使用最广泛的程序设计语言——C++进行了很好的介绍。本书从开篇就开始介绍现代C++程序设计技术，并介绍了大量关于如何使用C++标准库来简化程序设计的内容。

•适用于初学者以及任何希望学习新知识的人

本书主要是为那些从未编写过程序的人编写的，而且已经由超过1000名大学一年级新生试用过。不过，对于专业人员和高年级学生来说，通过观察公认的程序设计大师如何处理编程中的各种问题，同样也会获得新的领悟和指引。

•提供广阔的视野

本书第一部分非常广泛地介绍了基本程序设计技术，包括基本概念、设计和编程技术、语言特性以及标准库。这些内容教你如何编写具有输入、输出、计算以及简单图形显示等功能的程序。本书第二部分则介绍了一些更专门性的内容（如文本处理和测试），并提供了大量的参考资料。

本书网站 (<http://www.stroustrup.com/Programming/>) 提供了丰富的辅助资料，包括实例源码、PPT、勘误等。

作者简介

Bjarne Stroustrup 英国剑桥大学计算机科学博士，C++语言的设计者和最初的实现者，也是《C++程序设计语言》（已由机械工业出版社引进出版）一书的作者。他现在是德州农工大学计算机科学首席教授。1993年，由于在C++领域的重大贡献，Bjarne获得了ACM的Grace Murray Hopper大奖并成为ACM院士。在进入学术界之前，他曾在AT&T贝尔实验室工作多年。他是ISO C++标准委员会的创始人之一。



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



www.pearsonhighered.com

投稿热线: (010) 88379604
购书热线: (010) 68995259, 68995264
读者信箱: hzsj@hzbook.com



华章网站 <http://www.hzbook.com>

网上购书: www.china-pub.com



上架指导: 计算机 / 程序设计 / C++
ISBN 978-7-111-28248-8



9 787111 282488

定价: 89.00元



C++ 程序设计原理与实践

Programming

(美) Bjarne Stroustrup 著

Principles and Practice Using C++

英文版



机械工业出版社
China Machine Press

经典原版书库

C++ 程序设计原理与实践

Programming
Principles and Practice Using C++

(美) Bjarne Stroustrup 著



机械工业出版社
China Machine Press

English reprint edition copyright © 2010 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Programming: Principles and Practice Using C++* (ISBN 978-0-321-54372-1) by Bjarne Stroustrup, Copyright © 2009 Pearson Education, Inc.

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：01-2009-4962

图书在版编目（CIP）数据

C++程序设计原理与实践（英文版）/（美）斯特劳斯特鲁普（Stroustrup, B.）著.—北京：机械工业出版社，2009.10

（经典原版书库）

书名原文：Programming: Principles and Practice Using C++

ISBN 978-7-111-28248-8

I. C… II. 斯… III. C语言—程序设计—英文 IV. TP312

中国版本图书馆CIP数据核字（2009）第161265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

北京京师印务有限公司印刷

2009年10月第1版第1次印刷

150mm × 214mm · 39.625印张

标准书号：ISBN 978-7-111-28248-8

定价：89.00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

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅肇划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章分社较早意识到“出版要为教育服务”。自1998年开始，华章分社就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum, Bjarne Stroustrup, Brain W. Kernighan, Dennis Ritchie, Jim Gray, Afred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章分社欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzsj@hzbook.com
联系电话：(010) 88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037



Preface

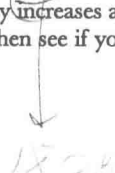
“Damn the torpedoes!
Full speed ahead.”

—Admiral Farragut

Programming is the art of expressing solutions to problems so that a computer can execute those solutions. Much of the effort in programming is spent finding and refining solutions. Often, a problem is only fully understood through the process of programming a solution for it.

This book is for someone who has never programmed before but is willing to work hard to learn. It helps you understand the principles and acquire the practical skills of programming using the C++ programming language. My aim is for you to gain sufficient knowledge and experience to perform simple useful programming tasks using the best up-to-date techniques. How long will that take? As part of a first-year university course, you can work through this book in a semester (assuming that you have a workload of four courses of average difficulty). If you work by yourself, don't expect to spend less time than that (maybe 15 hours a week for 14 weeks).

Three months may seem a long time, but there's a lot to learn and you'll be writing your first simple programs after about an hour. Also, all learning is gradual: each chapter introduces new useful concepts and illustrates them with examples inspired by real-world uses. Your ability to express ideas in code – getting a computer to do what you want it to do – gradually and steadily increases as you go along. I never say, “Learn a month's worth of theory and then see if you can use it.”



Why would you want to program? Our civilization runs on software. Without understanding software you are reduced to believing in “magic” and will be locked out of many of the most interesting, profitable, and socially useful technical fields of work. When I talk about programming, I think of the whole spectrum of computer programs from personal computer applications with GUIs (graphical user interfaces), through engineering calculations and embedded systems control applications (such as digital cameras, cars, and cell phones), to text manipulation applications as found in many humanities and business applications. Like mathematics, programming – when done well – is a valuable intellectual exercise that sharpens our ability to think. However, thanks to feedback from the computer, programming is more concrete than most forms of math, and therefore accessible to more people. It is a way to reach out and change the world – ideally for the better. Finally, programming can be great fun.

Why C++? You can't learn to program without a programming language, and C++ directly supports the key concepts and techniques used in real-world software. C++ is one of the most widely used programming languages, found in an unsurpassed range of application areas. You find C++ applications everywhere from the bottom of the oceans to the surface of Mars. C++ is precisely and comprehensively defined by a nonproprietary international standard. Quality and/or free implementations are available on every kind of computer. Most of the programming concepts that you will learn using C++ can be used directly in other languages, such as C, C#, Fortran, and Java. Finally, I simply like C++ as a language for writing elegant and efficient code.

This is not the easiest book on beginning programming; it is not meant to be. I just aim for it to be the easiest book from which you can learn the basics of real-world programming. That's quite an ambitious goal because much modern software relies on techniques considered advanced just a few years ago.

My fundamental assumption is that you want to write programs for the use of others, and to do so responsibly, providing a decent level of system quality; that is, I assume that you want to achieve a level of professionalism. Consequently, I chose the topics for this book to cover what is needed to get started with real-world programming, not just what is easy to teach and learn. If you need a technique to get basic work done right, I describe it, demonstrate concepts and language facilities needed to support the technique, provide exercises for it, and expect you to work on those exercises. If you just want to understand toy programs, you can get along with far less than I present. On the other hand, I won't waste your time with material of marginal practical importance. If an idea is explained here, it's because you'll almost certainly need it.

If your desire is to use the work of others without understanding how things are done and without adding significantly to the code yourself, this book is not for you. If so, please consider whether you would be better served by another book and another language. If that is approximately your view of programming, please also consider from where you got that view and whether it in fact is adequate for your needs. People often underestimate the complexity of program-

ming as well as its value. I would hate for you to acquire a dislike for programming because of a mismatch between what you need and the part of the software reality I describe. There are many parts of the “information technology” world that do not require knowledge of programming. This book is aimed to serve those who do want to write or understand nontrivial programs.

Because of its structure and practical aims, this book can also be used as a second book on programming for someone who already knows a bit of C++ or for someone who programs in another language and wants to learn C++. If you fit into one of those categories, I refrain from guessing how long it will take you to read this book, but I do encourage you to do many of the exercises. This will help you to counteract the common problem of writing programs in older, familiar styles rather than adopting newer techniques where these are more appropriate. If you have learned C++ in one of the more traditional ways, you’ll find something surprising and useful before you reach Chapter 7. Unless your name is Stroustrup, what I discuss here is not “your father’s C++.”

Programming is learned by writing programs. In this, programming is similar to other endeavors with a practical component. You cannot learn to swim, to play a musical instrument, or to drive a car just from reading a book – you must practice. Nor can you learn to program without reading and writing lots of code. This book focuses on code examples closely tied to explanatory text and diagrams. You need those to understand the ideals, concepts, and principles of programming and to master the language constructs used to express them. That’s essential, but by itself, it will not give you the practical skills of programming. For that, you need to do the exercises and get used to the tools for writing, compiling, and running programs. You need to make your own mistakes and learn to correct them. There is no substitute for writing code. Besides, that’s where the fun is!

On the other hand, there is more to programming – much more – than following a few rules and reading the manual. This book is emphatically not focused on “the syntax of C++.” Understanding the fundamental ideals, principles, and techniques is the essence of a good programmer. Only well-designed code has a chance of becoming part of a correct, reliable, and maintainable system. Also, “the fundamentals” are what last: they will still be essential after today’s languages and tools have evolved or been replaced.

What about computer science, software engineering, information technology, etc.? Is that all programming? Of course not! Programming is one of the fundamental topics that underlie everything in computer-related fields, and it has a natural place in a balanced course of computer science. I provide brief introductions to key concepts and techniques of algorithms, data structures, user interfaces, data processing, and software engineering. However, this book is not a substitute for a thorough and balanced study of those topics.

Code can be beautiful as well as useful. This book is written to help you see that, to understand what it means for code to be beautiful, and to help you to master the principles and acquire the practical skills to create such code. Good luck with programming!

A note to students

Of the 1000+ first-year students we have taught so far using drafts of this book at Texas A&M University, about 60% had programmed before and about 40% had never seen a line of code in their lives. Most succeeded, so you can do it, too.

You don't have to read this book as part of a course. I assume that the book will be widely used for self-study. However, whether you work your way through a part of a course or independently, try to work with others. Programming has an – unfair – reputation as a lonely activity. Most people work better and learn faster when they are part of a group with a common aim. Learning together and discussing problems with friends is not cheating! It is the most efficient – as well as most pleasant – way of making progress. If nothing else, working with friends forces you to articulate your ideas, which is just about the most efficient way of testing your understanding and making sure you remember. You don't actually have to personally discover the answer to every obscure language and programming environment problem. However, please don't cheat yourself by not doing the drills and a fair number of exercises (even if no teacher forces you to do them). Remember: programming is (among other things) a practical skill that you need to practice to master. If you don't write code (do several exercises for each chapter), reading this book will be a pointless theoretical exercise.

Most students – especially thoughtful good students – face times when they wonder whether their hard work is worthwhile. When (not if) this happens to you, take a break, reread the preface, and look at Chapter 1 (“Computers, People, and Programming”) and Chapter 22 (“Ideals and History”). There, I try to articulate what I find exciting about programming and why I consider it a crucial tool for making a positive contribution to the world. If you wonder about my teaching philosophy and general approach, have a look at Chapter 0 (“Notes to the Reader”).

You might find the weight of this book worrying, but it should reassure you that part of the reason for the heft is that I prefer to repeat an explanation or add an example rather than have you search for the one and only explanation. The other major part of the reason is that the second half of the book is reference material and “additional material” presented for you to explore only if you are interested in more information about a specific area of programming, such as embedded systems programming, text analysis, or numerical computation.

And please don't be too impatient. Learning any major new and valuable skill takes time and is worth it.

A note to teachers

No. This is not a traditional Computer Science 101 course. It is a book about how to construct working software. As such, it leaves out much of what a computer science student is traditionally exposed to (Turing completeness, state ma-

chines, discrete math, Chomsky grammars, etc.). Even hardware is ignored on the assumption that students have used computers in various ways since kindergarten. This book does not even try to mention most important CS topics. It is about programming (or more generally about how to develop software), and as such it goes into more detail about fewer topics than many traditional courses. It tries to do just one thing well, and computer science is not a one-course topic. If this book/course is used as part of a computer science, computer engineering, electrical engineering (many of our first students were EE majors), information science, or whatever program, I expect it to be taught alongside other courses as part of a well-rounded introduction.

Please read Chapter 0 (“Notes to the Reader”) for an explanation of my teaching philosophy, general approach, etc. Please try to convey those ideas to your students along the way.

Support

The book’s support website, www.stroustrup.com/Programming, contains a variety of materials supporting the teaching and learning of programming using this book. The material is likely to be improved with time, but for starters, you can find:

- Slides for lectures based on the book
- An instructor’s guide
- Header files and implementations of libraries used in the book
- Code for examples in the book
- Solutions to selected exercises
- Potentially useful links
- Errata

Suggestions for improvements are always welcome.

Acknowledgments

I’d especially like to thank my late colleague and co-teacher Lawrence “Pete” Petersen for encouraging me to tackle the task of teaching beginners long before I’d otherwise have felt comfortable doing that, and for supplying the practical teaching experience to make the course succeed. Without him, the first version of the course would have been a failure. We worked together on the first versions of the course for which this book was designed and together taught it repeatedly, learning from our experiences, improving the course and the book. My use of “we” in this book initially meant “Pete and me.”

Thanks to the students, teaching assistants, and peer teachers of ENGR 112 at Texas A&M University who directly and indirectly helped us construct this book, and to Walter Daugherty, who has also taught the course. Also thanks to Damian Dechev, Tracy Hammond, Arne Tolstrup Madsen, Gabriel Dos Reis, Nicholas Stroustrup, J. C. van Winkel, Greg Versoonder, Ronnie Ward, and Leor Zolman for constructive comments on drafts of this book. Thanks to Mogens Hansen for explaining about engine control software. Thanks to Al Aho, Stephen Edwards, Brian Kernighan, and Daisy Nguyen for helping me hide away from distractions to get writing done during the summers.

Thanks to the reviewers that Addison-Wesley found for me. Their comments, mostly based on teaching either C++ or Computer Science 101 at the college level, have been invaluable: Richard Enbody, David Gustafson, Ron McCarty, and K. Narayanaswamy. Also thanks to my editor, Peter Gordon, for many useful comments and (not least) for his patience. I'm very grateful to the production team assembled by Addison-Wesley; they added much to the quality of the book: Julie Grady (proofreader), Chris Keane (composer), Rob Mauhar (illustrator), Julie Nahil (production editor), and Barbara Wood (copy editor).

In addition to my own unsystematic code checking, Bashar Anabtawi, Yinan Fan, and Yuriy Solodkyy checked all code fragments using Microsoft C++ 7.1 (2003) and 8.0 (2005) and GCC 3.4.4.

I would also like to thank Brian Kernighan and Doug McIlroy for setting a very high standard for writing about programming, and Dennis Ritchie and Kristen Nygaard for providing valuable lessons in practical language design.

Contents

Preface *xxiii*

Chapter 0 Notes to the Reader 1

- 0.1 The structure of this book 2
 - 0.1.1 General approach 3
 - 0.1.2 Drills, exercises, etc. 4
 - 0.1.3 What comes after this book? 5
- 0.2 A philosophy of teaching and learning 6
 - 0.2.1 The order of topics 9
 - 0.2.2 Programming and programming language 10
 - 0.2.3 Portability 11
- 0.3 Programming and computer science 12
- 0.4 Creativity and problem solving 12
- 0.5 Request for feedback 12
- 0.6 References 13
- 0.7 Biographies 14
 - Bjarne Stroustrup 14
 - Lawrence "Pete" Petersen 15

xii Contents

Chapter 1 Computers, People, and Programming 17

- 1.1 Introduction 18
- 1.2 Software 19
- 1.3 People 21
- 1.4 Computer science 24
- 1.5 Computers are everywhere 25
 - 1.5.1 Screens and no screens 26
 - 1.5.2 Shipping 26
 - 1.5.3 Telecommunications 28
 - 1.5.4 Medicine 30
 - 1.5.5 Information 31
 - 1.5.6 A vertical view 32
 - 1.5.7 So what? 34
- 1.6 Ideals for programmers 34

Part I The Basics 41

Chapter 2 Hello, World! 43

- 2.1 Programs 44
- 2.2 The classic first program 45
- 2.3 Compilation 47
- 2.4 Linking 51
- 2.5 Programming environments 52

Chapter 3 Objects, Types, and Values 59

- 3.1 Input 60
- 3.2 Variables 62
- 3.3 Input and type 64
- 3.4 Operations and operators 66
- 3.5 Assignment and initialization 69
 - 3.5.1 An example: detect repeated words 71
- 3.6 Composite assignment operators 73
 - 3.6.1 An example: find repeated words 73
- 3.7 Names 74
- 3.8 Types and objects 77
- 3.9 Type safety 78
 - 3.9.1 Safe conversions 79
 - 3.9.2 Unsafe conversions 80

Chapter 4 Computation 89

- 4.1 Computation 90
- 4.2 Objectives and tools 92

4.3	Expressions	94
4.3.1	Constant expressions	95
4.3.2	Operators	96
4.3.3	Conversions	98
4.4	Statements	99
4.4.1	Selection	101
4.4.2	Iteration	108
4.5	Functions	112
4.5.1	Why bother with functions?	114
4.5.2	Function declarations	115
4.6	Vector	116
4.6.1	Growing a vector	118
4.6.2	A numeric example	119
4.6.3	A text example	121
4.7	Language features	123
Chapter 5	Errors	131
5.1	Introduction	132
5.2	Sources of errors	134
5.3	Compile-time errors	134
5.3.1	Syntax errors	135
5.3.2	Type errors	136
5.3.3	Non-errors	137
5.4	Link-time errors	137
5.5	Run-time errors	138
5.5.1	The caller deals with errors	140
5.5.2	The callee deals with errors	141
5.5.3	Error reporting	143
5.6	Exceptions	144
5.6.1	Bad arguments	145
5.6.2	Range errors	146
5.6.3	Bad input	148
5.6.4	Narrowing errors	151
5.7	Logic errors	152
5.8	Estimation	155
5.9	Debugging	156
5.9.1	Practical debug advice	157
5.10	Pre- and post-conditions	161
5.10.1	Post-conditions	163
5.11	Testing	164

xiv Contents

Chapter 6 Writing a Program	171
6.1 A problem	172
6.2 Thinking about the problem	173
6.2.1 Stages of development	174
6.2.2 Strategy	174
6.3 Back to the calculator!	176
6.3.1 First attempt	177
6.3.2 Tokens	179
6.3.3 Implementing tokens	181
6.3.4 Using tokens	183
6.3.5 Back to the drawing board	185
6.4 Grammars	186
6.4.1 A detour: English grammar	191
6.4.2 Writing a grammar	192
6.5 Turning a grammar into code	193
6.5.1 Implementing grammar rules	194
6.5.2 Expressions	195
6.5.3 Terms	198
6.5.4 Primary expressions	200
6.6 Trying the first version	201
6.7 Trying the second version	206
6.8 Token streams	207
6.8.1 Implementing <code>Token_stream</code>	209
6.8.2 Reading tokens	211
6.8.3 Reading numbers	212
6.9 Program structure	213
Chapter 7 Completing a Program	219
7.1 Introduction	220
7.2 Input and output	220
7.3 Error handling	222
7.4 Negative numbers	227
7.5 Remainder: %	228
7.6 Cleaning up the code	231
7.6.1 Symbolic constants	231
7.6.2 Use of functions	233
7.6.3 Code layout	234
7.6.4 Commenting	236
7.7 Recovering from errors	238
7.8 Variables	241
7.8.1 Variables and definitions	241
7.8.2 Introducing names	246
7.8.3 Predefined names	249
7.8.4 Are we there yet?	249