C和C++代码锦囊
——实用开发者指南（影印版）
C&C++ CODE CAPSULES
A GUIDE FOR PRACTITIONERS

（美）Chuck Allison 编著

- Real solutions for real programmers
- 100% ISO Compliant
- Supercharge Productivity with the Standard C and C++ libraries
- From Pointers to templates - and beyond
- Based on Chuck Allison's respected **C/C++ Users Journal column**

Pearson Education

科 学 出 版 社

C 和 C++代码锦囊
——实用开发者指南

（影印版）

C & C++ Code Capsules
A Guide For Practitioners

（美） Chuck Allison 编著

科 学 出 版 社

北 京

图字：01-2003-6794 号

内 容 简 介

本书指导读者如何充分利用标准 C 和 C++库，其涵盖容器、迭代器、算法、文本与文件处理、时间与日期处理以及内存管理等内容。同时本书提供了非常实用的有关抽象、模板、二进制处理、可见度、控制结构和异常处理的使用技巧。

本书还提供了 C 和 C++的大量编程范例，对于进行实用开发的人员具有很大的借鉴和指导意义。各类读者均能从中获得裨益，提高 C/C++的编程水平。

图书在版编目(CIP)数据

C & C++ 代码锦囊：实用开发者指南= C & C++ Code Capsules: A Guide for Practitioners / (美) Allison, C. 著.—影印本. —北京：科学出版社，2004

ISBN 7-03-012469-3

Ⅰ.C… Ⅱ.A… Ⅲ.C 语言—程序设计—英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 099957 号

策划编辑：李佩乾/责任编辑：朱凤成

责任印制：吕春珉/封面制作：北新华文平面设计室

科 学 出 版 社 出版

北京东黄城根北街16号

邮政编码：100717

http://www.sciencep.com

双 青 印 刷 厂 印刷

科学出版社发行 各地新华书店经销

*

2004 年 1 月第 一 版 开本：787×960 1/16

2004 年 1 月第一次印刷 印张：36 3/4

印数：1—3 000 字数：588 000

定价：60.00 元

(如有印装质量问题，我社负责调换<环伟>)

Foreword

Bruce Eckel

I first heard about Chuck when his bit handling classes, bitset and bitstring, were accepted into the standard C++ library in March 1993 (bitstring was later absorbed when the STL was added). While I myself did add a couple of tiny bits to the C++ standard here and there, the idea of successfully running an entire class through the terrible gauntlet of the ANSI/ISO C++ committee (on which we were both participants, so we knew) impressed me greatly.

But in the world of computers, which is so full of overstatement that I must ironically rely on my intuition rather than my intellect to discern truth, what impresses me more is someone who can explain things in a simple, clear, and non-overblown way. That is, a great teacher. Chuck is a great teacher. You can see it in the activities he finds himself compelled to do: writing, teaching, editing, explaining. When I find someone like this—and I know for sure when I see them speak to an audience—I convince them to speak at the Software Development (SD) conference (where I chair the C++ and Java tracks). Chuck has become a regular fixture at the conference, satisfying audiences on both coasts.

At the last SD conference (Fall 1997 in Washington, DC), it was Chuck's birthday, and when we found this out a group of us took him out to dinner. It was only after we were finally seated that I looked around and realized we were all authors: Bjarne Stroustrup (creator of C++ and author of *The C++ Programming Language*), Dan Saks (C++ columnist, speaker, consultant, and long-time secretary of the ANSI/ISO C++ committee), Bobby Schmidt (CUJ columnist, speaker), Marco Cantu (author of the "Mastering Delphi" books as well as C++ books), Tim Gooch (editor of the Cobb Group publications on C++ and now Java), and myself. These are the folks who respect Chuck enough to buy him dinner.

There are, of course, lots of "introduction to C++" books. Sometimes I feel like I keep trying to write a book on that subject over and over (my—I hope—final effort was *Thinking in C++*). But what happens when you've understood the basics and you want more depth? Books exist, but they can often be written in the tongue of the experts (a language that leaves me gasping) or they cover topics that are too esoteric or advanced. This book provides a bridge to the world of advanced topics; it gives you what you need but it won't overwhelm you in the process.

Chuck has made his book both clear and accurate, and accuracy is something else I'm exceptionally fond of. When a book has too many flaws I grow tired of it (in the early days we had to put up with such things, but now there are enough carefully created C++ books that there's no reason to waste your time). Another thing I like a lot about this book is the brevity of the chapters, and the way each one is focused on a single topic, so I can pick it up and get an entire concept at once (I have a somewhat short attention span). This is a book that you will enjoy over time as it hands you one insight after another.

Bruce Eckel
http://www.EckelObjects.com
October 1997

Preface

This book is for people who program in C and C++ for a living. It assumes that you are already familiar with the syntax and basic constructs of both languages, and it offers practical wisdom for creating effective, real-world programs. Each code capsule, or sample program, contains working code illustrating proven idioms and techniques that leverage the power of these important programming languages.

This book serves as a voice of experience for those who wish to strengthen their skills and improve their effectiveness in the workplace. Despite current fervor for the object-oriented paradigm (which this book abundantly embraces), I make no excuse for paying homage to the C foundations of C++. I have found too many developers ill-prepared to master C++ because they lack a thorough understanding of basic concepts such as pointers, scope, linkage, and static type checking. Perhaps the biggest deficiency of all is a lack of familiarity with the standard C library. It is sad indeed when developers waste time reinventing what the library functions already provide so well. The C++ novice is often too eager to abandon (i.e., gloss over) simple C in favor of the "exciting" features of C++, such as inheritance, exceptions, or overriding operator new, even when such are not warranted. I feel confident that everyone will learn something from these pages. Chapters 1 and 13 through 16 are strictly C++ chapters, and Chapters 4 through 6 apply only to the C language. All other chapters cover both the C and C++ aspects of their respective topic.

That said, this is primarily a C++ book. As it goes to press, the C++ standardization effort is in its home stretch. The second public committee draft (CD2) has completed its cycle and only minor edits remain. As a member of this committee since early 1991, I have seen its document grow from 200 to over 750 pages. We have added exceptions, templates, namespaces, runtime type identification (RTTI) and other features to the language, and a sophisticated, templatized system of interrelated algorithms, containers, and iteration constructs to the library (commonly known as the Standard Template Library, or STL). Unlike other standards efforts, this committee has concentrated as much on invention as on standardizing existing practice. The overwhelming intricacies of C++ caused one Internet surfer to post this message: "If C gives you enough rope to hang yourself, then C++ gives you enough rope to hang everyone in your neighborhood, hoist the riggings of a small sailing ship, and still have enough left over to hang yourself." I have labored to illustrate and motivate standard C++ and its library in such a way that you might use your rope more wisely.

The first chapter (Chapter 0), an excerpt from an interview I conducted with Bjarne Stroustrup, records his feelings about the state of C++ as it becomes a standard. The rest of the book is divided into three parts.

Part I: Preliminaries

After a brief tour of C++, these chapters close some of the gaps a typical C programmer might have before s/he prepares to tackle C++. Chapter 2, "Pointers," is based on a well-received three-part series I ran in the *C Users Journal* in 1993. Chapters 4 through 6 cover what every professional should know about the standard C library, which is a crucial part of standard C++.

Part II: Key Concepts

This section thoroughly motivates and illustrates the concepts and features of the C++ language. Chapter 7 introduces data abstraction through classes, and Chapter 8 covers type abstraction as implemented by the C++ template mechanism. Templates are every bit as crucial to the effective use of C++ as objects are, perhaps even more so. Chapter 14 not only treats inheritance and polymorphism, but also illustrates object-oriented design and reuse as it presents a framework for object persistence that works with today's relational database management systems. The chapters in between give the reader depth in important fundamental concepts that too many developers tend to overlook.

Part III: Leveraging the Standard Library

Chapters 15 through 20 show how to use and appreciate the notable components of the standard C++ library, as well as elucidate some of the more sophisticated features of the standard C library that went beyond the scope of Chapters 4, 5, and 6. Chapters 15 and 16 explain why the STL subset of the library is what it is, and how to use it effectively. Chapter 19 contains a useful date component that can even handle partial dates, a common business data processing requirement.

In summary, this is book about what works. I've attempted to steer the reader away from the "gotchas" by illustrating "best practices" with a reasonable balance of breadth and depth. Why another C++ book in 1998? Because the language and library haven't stabilized until now. This book goes to press just one week after the standards committee met to approve the final draft of ISO C++, and I have taken care to steer clear of any dark corners that remain (all languages and environment have them). I am confident that all the material in this book will be timely for years to come.

Acknowledgments

This book has actually been in the making for more years than I care to admit. It began in 1984 when I chaired the Computer Science department at Pima Community College in Tucson, Arizona, and my colleague Claire Hamlet persuaded me to pursue a grant to develop the first course in C programming there. Thereafter I started collecting C program examples and shared them with my fellow employees at Hughes Aircraft Company and at the World Headquarters of the Church of Jesus Christ of Latter-day Saints in Salt Lake City. The "code capsule" idea, short vignettes with examples on particular topics, grew from my effort to make learning C fun and relatively painless for COBOL refugees. I was fortunate enough for a time to have management support in chairing an internal C Language Support Committee, a first-rate team of experienced programmers (David Coombs, John Pearson, Lorin Lords, Kent Olsen, Bill Owens, Drew Terry, and Mike Terrazas) that developed an outstanding curriculum and effectively trained over 100 Church employees.

The name "code capsules" occurred to me over breakfast with Lee Copeland in the Church cafeteria (he has a way of keeping me on my toes). Mike Terrazas reviewed early versions of these vignettes, and suggested that I present them to the *C Users Journal*, which I unwittingly did by showing them to P. J. "Bill" Plauger in London at the March 1992 meeting of the C++ Standards Committee. As senior editor, he proposed that I become a columnist for the *Journal*. The *Code Capsule* column ran from October 1992 until May 1995, when time commitments forced me to resign. A valued mentor and friend, Bill has also been instrumental in encouraging me to put these capsules in book form. Bruce Eckel was very kind to review portions of the book, and Pete Becker scoured the entire manuscript, uncovering a number of errors and inconsistencies. When it comes to "support," however, that necessary intangible that keeps one going, I must follow the example of almost every author who has ever written by thanking my family. Only Sandy, James, and Kim have a feel for the magnitude of the effort that has brought these pages to press. As 21-year-old James recently wrote from England, where he is spending two years, "So, Dad is finally finishing his book! He's been working on that for as long as I can remember."

Chuck Allison
http://www.freshsources.com
Novemver 1997

Contents

Foreword ix

Bruce Eckel

Preface xi

C++: The Making of a Standard 1

An Interview with Bjarne Stroustrup

Part I Preliminaries 9

1 A Better C 11

A Tale Of Two Languages 11
Incremental Journey 12
The Type System 12
Function Prototypes 13
Type-safe Linkage 17
References 19
Type-safe I/O 20
Standard Streams 21
Formatting 24
Manipulators 29
Function Overloading and Function Templates 32
Operator Overloading 32
Inline Functions 34
Default Arguments 35
`new` and `delete` 36

Statement Declarations 37
Standard Library Features 38
C Compatibility 38
Summary 40

2 Pointers 40

Programming on the Edge 40
The Basics 40
Pointer Arithmetic 44
Pass-By-Reference Semantics 48
Generic Pointers 49
`const` Pointers 50
Pointers and One-Dimensional Arrays 52
Arrays as Parameters 56
Arrays of Strings 59
Pointers and Multidimensional Arrays 62
Higher and Deeper 64
Pointers to Functions 68
Pointers to Member Functions 71
Encapsulation and Incomplete Types 73
Summary 78

3 The Preprocessor 79

The `#include` Directive 79
Other Preprocessor Directives 80
Predefined Macros 83
Conditional Compilation 85
Preprocessing Operators 86
Implementing assert 88
Macro Magic 89
Character Sets, Trigraphs, and Digraphs 93
Phases of Translation 96
Summary 97

4 The Standard C Library, Part I 98
For the Adequate Programmer

`<ctype.h>` 99
`<stdio.h>` 102
`<stdlib.h>` 109
`<string.h>` 115

5 The Standard C Library, Part II 118

For the Polished Programmer

<assert.h> 118
<limits.h> 119
<stddef.h> 122
<time.h> 125
Appendix 5.1: Character Sets 127
Code Pages 129
Character Set Standards 130
ISO 10646 130
Unicode 131

6 The Standard C Library, Part III 131

For the Complete Programmer

<float.h> 132
<math.h> 132
<errno.h> 138
<locale.h> 138
<setjmp.h> 140
<signal.h> 142
<stdarg.h> 142
va_list's As Arguments 143
An Application 146
Conclusion 150
Appendix 6.1: Floating-point Number Systems 151

Part II Key Concepts 159

7 Abstraction 161

Data Abstraction 161
Operator Overloading 168
Concrete Data Types 173
Type Abstraction 180
Function Abstraction 181
Summary 184

8 Templates 185

Generic Programming 187
Function Templates 187

Class Templates 189
Template Parameters 194
Template Specialization 196
Summary 199

9 Bit Manipulation 202

Bitwise Operators 202
Accessing Individual Bits 204
Large Bitsets 209
Bit Strings 224
Wish List 224
The `bitset` Template 224
The `vector<bool>` Template Specialization 227
Summary 228

10 Conversions and Casts 229

Integral Promotion 229
Demotions 233
Arithmetic Conversions 235
Function Prototypes 237
Explicit Conversions 239
Function-style Casts 240
Const Correctness 240
User-Defined Conversions 243
Beefing up `operator[]` 249
New-Style Casts 252
Summary 253

11 Visibility 255

What's In a Name? 255
Scope 255
Minimal Scoping 258
Class Scope 259
Nested Classes 265
Local Classes 266
Classic Namespaces 268
Namespace Scope 269
Lifetime 272
Lifetime of Temporaries 275
Linkage 276
Type-safe Linkage 280

"Language" Linkage 281
Summary 281

12 Control Structures 283

Structured Programming 283
Branching 290
Nonlocal Branching 296
Signals 300
Summary 305
References 305

13 Exceptions 306

Error Handling Alternatives 306
Stack Unwinding 314
Catching Exceptions 316
Standard Exceptions 318
Resource Management 322
Constructors and Exceptions 322
Memory Management 330
Exception Specifications 331
An Error-Handling Strategy 333
Summary 337

14 Object-oriented Programming 338

Inheritance 342
Heterogeneous Collections 343
Virtual Functions and Polymorphism 344
Abstract Base Classes 347
Case Study: A Framework for Object Persistence 350
Database Access 353
Mapping Objects to Relational Schema 354
PFX Architecture 356
A Code Walkthrough 359
Summary 379

Part III Leveraging the Standard Library 381

15 Algorithms 383

Complexity 386
Generic Algorithms 388

Function Objects 392
Function Taxonomy 394
Function Object Adapters 395
Algorithm Taxonomy 396
Summary 399
References 399

16 Containers and Iterators 398

Standard Containers 401
Iterators 402
Iterator Taxonomy 403
Special-Purpose Iterators 405
Container Adapters 408
Associative Containers 409
Applications 409
Non-STL Containers 417
Summary 419

17 Text Processing 421

scanf 421
printf 427
Substrings 429
The Standard C++ String Class 436
String Streams 440
Wide Strings 441
Summary 442

18 File Processing 443

Filters 443
Binary Files 446
Record Processing 448
Temporary Files 454
Portability 455
POSIX 455
File Descriptors 456
Copying Files via Descriptors 456
Reading Directory Entries 458
Redirecting Standard Error 462
Encapsulating Directory Operations 466
Summary 471

19 Time and Date Processing 472
Julian Day Encoding 475
Date Classes for Real Work 495
Calculating the Week of the Year 521
Summary 522
Reference 522

20 Dynamic Memory Management 523
Ragged Arrays 523
Using the Heap in Standard C 525
The C++ Free Store 529
Deep versus Shallow Copy 530
Handling Memory Allocation Failure 533
Overriding `new` and `delete` 534
Placement `new` 535
Managing the Heap 538
Avoiding Memory Management 539
Summary 546

Appendices **547**

A C/C++ Compatibility 547

B Standard C++ Algorithms 549

C Function Objects and Adapters 558
Function Objects 558
Function Object Adapters 560

D Annotated Bibliography 561
The C Practitioner's Booklist 561
The C++ Practitioner's Booklist 562

Index **565**

C++: The Making of a Standard

(An Interview with Bjarne Stroustrup)

As this book goes to press, the second official committee draft (CD2) has been released, and national standards bodies have given their comments to the joint ISO/ANSI standards committee (X3J16/WG21). The features of the language have been stable for some time. Shortly before the committee met to approve CD2 in Stockholm in July 1996, I had the opportunity to interview Bjarne Stroustrup for the *C/C++ Users Journal.* I was interested in getting his feelings on the state of the language and his thoughts on the future of C++. This chapter is an excerpt from that interview.

To appreciate the work that has gone into the standard, as well as the interview that follows, a little history is in order. For a detailed technical and anecdotal history of C++, see Stroustrup's *The Design and Evolution of C++* (Addison-Wesley, 1994).

Bjarne Stroustrup, a Dane with a Ph.D. from Cambridge University (England), had used the Simula language for distributed systems simulations in his research. He was disappointed with its poor performance, however, so in 1979 when his new employer, AT&T Bell Labs, invited him to "do something interesting," he decided to infuse the C language with some Simula features he had grown accustomed to—most notably classes. Thus C with Classes was born. It caught on at AT&T, was dubbed "C++," and then proceeded to become a support burden for its inventor. After the first edition of Stroustrup's *The C++ Programming Language* (Addison-Wesley, 1985) hit the shelves, however, there was no turning back; the language became too popular. And as you might expect, multiple implementations appeared, each with its own special features. About the time the C standard became official, the major players in the C++ community were pushing for a C++ standard. ANSI committee X3J16 met for the first time in December 1989, with Dmitry Lenkov of Hewlett-Packard as chair. Steve Clamage of Sun Microsystems, himself an early implementor of C++, became chair in 1996.

The base documents for the committee included the ISO C standard, as well as Ellis and Stroustrup's *The Annotated C++ Reference Manual* (ARM). The latter pretty much reflected

AT&T C++ 2.0, along with Bjarne's ideas for extensions (mainly templates and exceptions). The main goals at the outset were to standardize IOStreams, and to add templates and exceptions to the language. Due to the large number of non-U.S. participants, the committee voted to work jointly with ISO Working Group 21 beginning at the June 1991 meeting in Lund, Sweden. It looked like things were winding down at the end of 1993, yet there was still some uneasiness about a lack of robustness in the standard library. At the San Jose meeting in November 1993, Alex Stepanov, then of Hewlett-Packard, gave a presentation on generic programming that really put templates to good use. By the San Diego meeting the next March, he had refined his and Meng Lee's standard template library (STL) to the point that the committee was ready to consider it seriously, even if it meant delaying the completion of the standard. I remember being one of the conservative skeptics at the time, yet my hand went up in favor of STL, which, along with IOStreams, is now the centerpiece of the standard C++ library.

The major features added since C++ 2.0 include templates, exceptions, user namespaces, and runtime type identification (RTTI). They are "major" because they directly affect the overall structure of your programs. Minor features, which are less intrusive but very powerful in their own right, include new-style casts, a new boolean data type (`bool`), a capacity to overload on enumerated types, support for wide characters, and alternative tokens to support foreign keyboards (e.g., `or` for `||`). In addition to STL, the standard library includes renovated stream classes, a string class template with specializations for wide and narrow characters, and the infrastructure for RTTI and for overloading and overriding `operator new` and `operator delete`.

Chuck Allison: I know you have a Ph.D. in Applied Math. What were your other degrees?

Bjarne Stroustrup: Not quite; my Masters degree (Cand.Scient.) from the University of Aarhus (Denmark) is in "Mathematics with computer science." I took the math part because that was the only way to do computer science there in those days. My Ph.D. from Cambridge (England, of course) is in Computer Science. I'm a very poor mathematician, but I guess that's better than not being one at all.

CA: How did you get into computing?

BS: By signing up for math and computer science in university. I have tried to remember why I did that, but I really don't know. I certainly hadn't seen a computer by the time I signed up. I guess the combination of scientific and practical aspects attracted me.

CA: When did you notice that C with Classes was becoming something that was going to consume much of your time, if not your career, and not just a temporary interest?

BS: Somewhere in 1982 I realized that support of the C with Classes user community was becoming an unacceptable drain on me. As I saw it, the user population had become too large for one person to serve well while also doing research, but too small to support an infrastructure. I had to decide whether to develop C with Classes into a more flexible and expressive