

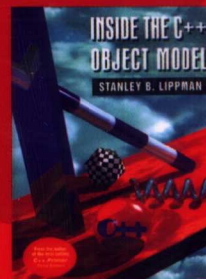
原版风暴 · 深入 C++ 系列



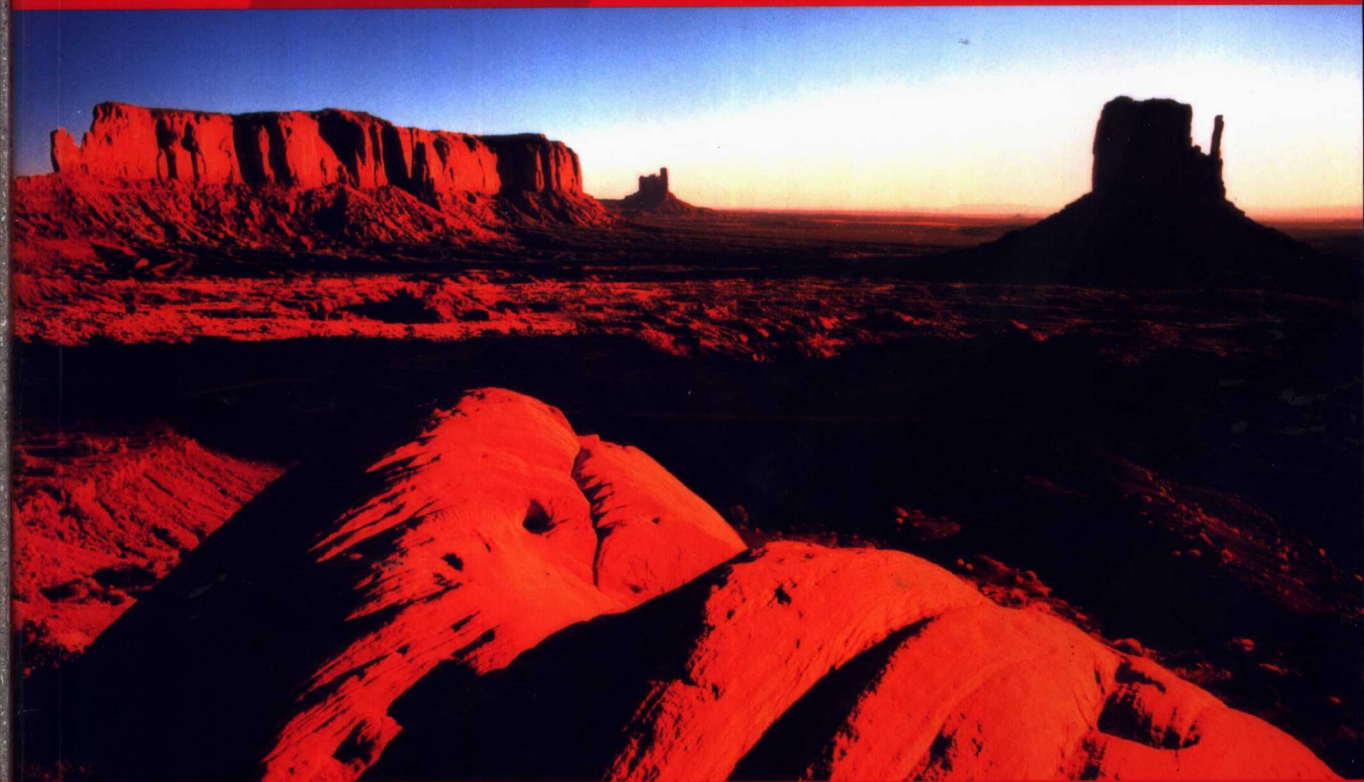
Inside the C++ Object Model

# 深度探索 C++ 对象模型

(影印版)



[美] Stanley B. Lippman 著



中国电力出版社  
www.infopower.com.cn

原版风暴 · 深入 C++ 系列

**Inside the C++ Object Model**

**深度探索 C++  
对象模型**

(影印版)

[美] Stanley B. Lippman 著

中国电力出版社

Inside The C++ Object Model. (ISBN 0-201-83454-5)

Stanley B. Lippman

Copyright © 1996 by Addison Wesley Longman , Inc.

Original English Language Edition Published by Addison Wesley Longman , Inc.

All rights reserved.

Reprinting edition published by PEARSON EDUCATION ASIA LTD and CHINA ELECTRIC POWER PRESS, Copyright © 2003.

本书影印版由 Pearson Education 授权中国电力出版社在中国境内（香港、澳门特别行政区和台湾地区除外）独家出版、发行。

未经出版者书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有 Pearson Education 防伪标签，无标签者不得销售。

北京市版权局著作合同登记号：图字：01-2003-4142

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

### 图书在版编目（CIP）数据

深度探索 C++对象模型 / (美) 李普曼著. —影印本. —北京：中国电力出版社，2003  
(原版风暴·C++系列)

ISBN 7-5083-1405-0

I.深... II.李... 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 048301 号

**责任编辑：朱恩从**

**丛书名：原版风暴·C++系列**

**书名：深度探索C++对象模型（影印版）**

**编著：（美）Stanley B. Lippman**

**出版发行：中国电力出版社**

地址：北京市三里河路6号 邮政编码：100044

电话：（010）88515918 传真：（010）88518169

**印刷：北京地矿印刷厂**

**发行者：新华书店总店北京发行所**

**开本：787×1092 1/16 印 张：19**

**书 号：ISBN 7-5083-1405-0**

**版 次：2003年8月北京第一版**

**印 次：2003年8月第一次印刷**

**定 价：42.00 元**

***for Beth***

who suffered with forbearance  
the widowhood of an author's wife  
and nonetheless provided wildly unreasonable support

***for Danny & Anna***

who suffered with far less forbearance  
but no less love

# Preface

For nearly a decade within Bell Laboratories, I labored at implementing C++. First it was on *cfront*, Bjarne Stroustrup's original C++ implementation (from Release 1.1 back in 1986 through Release 3.0, made available in September 1991). Then it was on to what became known internally as the *Simplifier*, the C++ Object Model component of the Foundation project. It was during the *Simplifier*'s design period that I conceived of and began working on this book.

What was the Foundation project? Under Bjarne's leadership, a small group of us within Bell Laboratories was exploring solutions to the problems of large-scale programming using C++. The Foundation was an effort to define a new development model for the construction of large systems (again, using C++ only; we weren't providing a multilingual solution). It was an exciting project, both for the work we were doing and for the people doing the work: Bjarne, Andy Koenig, Rob Murray, Martin Carroll, Judy Ward, Steve Buroff, Peter Juhl, and myself. Barbara Moo was supervising the gang of us other than Bjarne and Andy. Barbara used to say that managing a software group was like herding a pride of cats.

We thought of the Foundation as a kernel upon which others would layer an actual development environment for users, tailoring it to a UNIX or Smalltalk model as desired. Internally, we called it Grail, as in the quest for, etc. (It seems a Bell Laboratories tradition to mock one's most serious intentions.)

Grail provided for a persistent, semantic-based representation of the program using an object-oriented hierarchy Rob Murray developed and named ALF. Within Grail, the traditional compiler was factored into separate executables. The parser built up the ALF representation. Each of the other components (type checking, simplification, and code generation) and any tools, such as a browser, operated on (and possibly augmented) a centrally stored ALF representation of the program. The *Simplifier* is the part of the compiler between type checking and code generation. (Bjarne came up with the name *Simplifier*; it is a phase of the original *cfront* implementation.)

What does a Simplifier do between type checking and code generation? It transforms the internal program representation. There are three general flavors of transformations required by any object model component:

1. *Implementation-dependent transformations.* These are implementation-specific aspects and vary across compilers. Under ALF, they involved the transformations of what we called “tentative” nodes. For example, when the parser sees the expression

```
fct();
```

it doesn't know if this is (a) an invocation of a function represented or pointed to by `fct` or (b) the application of an overloaded call operator on a class object `fct`. By default, the expression is represented as a function call. The Simplifier rewrites and replaces the call subtree when case (b) applies.

2. *Language semantics transformations.* These include constructor/destructor synthesis and augmentation, memberwise initialization and memberwise copy support, and the insertion within program code of conversion operators, temporaries, and constructor/destructor calls.
3. *Code and object model transformations.* These include support for virtual functions, virtual base classes and inheritance in general, operators **new** and **delete**, arrays of class objects, local static class instances, and the static initialization of global objects with non-constant expressions. An implementation goal I aimed for in the Simplifier was to provide an Object Model hierarchy in which the object implementation was a virtual interface supporting multiple object models.

These last two categories of transformations form the basis of this book. Does this mean this book is written for compiler writers? No, absolutely not. It is written by a (former) compiler writer (that's me) for intermediate to advanced C++ programmers (ideally, that's you). The assumption behind this book is that the programmer, by understanding the underlying C++ Object Model, can write programs that are both less error prone and more efficient.

## What Is the C++ Object Model?

There are two aspects to the C++ Object Model:

1. The direct support for object-oriented programming provided within the language
2. The underlying mechanisms by which this support is implemented

The language level support is pretty well covered in my *C++ Primer* and in other books on C++. The second aspect is barely touched on in any current text, with the exception of brief discussions within [ELLIS90] and [STROUP94]. It is this second aspect of the C++ Object Model that is the primary focus of this book. (In that sense, I consider this text to form a book-end to my *C++ Primer*, much as my MFA and MS degrees provide a “fearful symmetry” to my education.) The language covered within the text is the draft Standard C++ as of the winter 1995 meeting of the committee. (Except for some minor details, this should reflect the final form of the language.)

The first aspect of the C++ Object Model is invariant. For example, under C++ the complete set of virtual functions available to a class is fixed at compile time; the programmer cannot add to or replace a member of that set dynamically at runtime. This allows for extremely fast dispatch of a virtual invocation, although at the cost of runtime flexibility.

The underlying mechanisms by which to implement the Object Model are not prescribed by the language, although the semantics of the Object Model itself make some implementations more natural than others. Virtual function calls, for example, are generally resolved through an indexing into a table holding the address of the virtual functions. Must such a virtual table be used? No. An implementation is free to introduce an alternative mechanism. Moreover, if a virtual table is used, its layout, method of access, time of creation, and the other hundred details that must be decided, are all decisions left to each implementation. Having said that, however, I must also say that the general pattern of virtual function implementation across all current compilation systems is to use a class-specific virtual table of a fixed size that is constructed prior to program execution.

If the underlying mechanisms by which the C++ Object Model is implemented are not standardized, then one might ask, why bother to discuss them at all? The primary reason is because my experience has shown that if a programmer understands the underlying implementation model, the

programmer can code more efficiently and with greater confidence. Determining when to provide a copy constructor, and when not, is not something one should guess at or have adjudicated by some language guru. It should come from an understanding of the Object Model.

A second reason for writing this book is to dispel the various misunderstandings surrounding C++ and its support of object-oriented programming. For example, here is an excerpt from a letter I received from someone wishing to introduce C++ into his programming environment:

I work with a couple of individuals who have not written and/or are completely unfamiliar with C++ and OO. One of the engineers who has been writing C code since 1985 feels very strongly that C++ is good only for user-type applications, but not server applications. What he is saying is to have a fast and efficient database level engine that it must be written in C compared to C++. He has identified that C++ is bulky and slow.

C++, of course, is not inherently bulky and slow, although I've found this to be a common assumption among many C programmers. However, just saying that is not very convincing, particularly if the person saying it is perceived as a C++ partisan. This book is partially an attempt to lay out as precisely as I can the kinds of overhead that are and are not inherent in the various Object facilities such as inheritance, virtual functions, and pointers to class members.

Rather than answering the individual myself, I forwarded his letter to Steve Vinoski of Hewlett-Packard, with whom I had previously corresponded regarding the efficiency of C++. Here is an excerpt from his response:

I have heard a number of people over the years voice opinions similar to those of your colleagues. In every case, those opinions could be attributed to a lack of factual knowledge about the C++ language. Just last week I was chatting with an acquaintance who happens to work for an IC testing manufacturer, and he said they don't use C++ because "it does things behind your back." When I pressed him, he said that he understood that C++ calls `malloc()` and `free()` without the programmer knowing it. This is of course not true. It is this sort of "myth and legend" that leads to opinions such as those held by your colleagues....

Finding the right balance [between abstraction and pragmatism] requires knowledge, experience, and above all, thought. Using C++ well requires effort, but in my experience the returns on the invested effort can be quite high.



I like to think of this book, then, as my answer to this individual, and, I hope, a repository of knowledge to help put to rest many of the myths and legends surrounding C++.

If the underlying mechanisms supporting the C++ Object Model vary both across implementations and over time, how can I possibly provide a general discussion of interest to any particular individual? Static initialization provides an interesting case in point.

Given a class *X* with a constructor, such as the following:

```
class X
{
    friend istream&
        operator>>( istream&, X& );
public:
    X( int sz = 1024 ) { ptr = new char[ sz ]; }
    ...
private:
    char *ptr;
};
```

and the declaration of a global object of class *X*, such as the following:

```
X buf;

int main()
{
    // buf must be constructed at this point
    cin >> setw( 1024 ) >> buf;
    ...
}
```

the C++ Object Model guarantees that the *X* constructor is applied to *buf* prior to the first user statement of *main()*. It does not, however, prescribe how that is to get done. The solution is called *static initialization*; the actual implementation depends on the degree of support provided by the environment.

The original *cfront* implementation not only presumed no environment support. It also presumed no explicit platform target. The only presumption was that of being under some variant of UNIX. Our solution, therefore, was specific only to UNIX: the presence of the **nm** command. The **CC** command (a UNIX shell script for portability) generated an executable, ran the **nm** command on the executable—thereby generating a new *.c* file—compiled the *.c* file, and then relinked the executable. (This was called the *munch*

solution.) This did the job by trading compile-time efficiency for portability. Eventually, however, users chafed under the compile-time overhead.

The next step was to provide a platform-specific solution: a COFF-based program (referred to as the *patch* solution) that directly examined and threaded the program executable, thus doing away with the need to run `nm`, `compile`, and `relink`. (COFF was the Common Object File Format for System V pre-Release 4 UNIX systems.) Both of these solutions are program-based, that is, within each `.c` file requiring static initialization `cfront` generated an `sti` function to perform the required initializations. Both `munch` and `patch` solutions searched for functions bearing an `sti` prefix and arranged for them to be executed in some undefined order by a `_main()` library function inserted as the first statement of `main()`.

In parallel with these releases of `cfront`, a System V COFF-specific C++ compiler was under development. Targeted for a specific platform and operating system, this compiler was able to effect a change in the System V link editor: a new initialize section that provided for the collection of objects needing static initialization. This extension of the link editor provides what I call an *environment-based* solution that is certainly superior to a program-based solution.

So any generalization based on the `cfront` program-based solution would be misleading. Why? Because as C++ has become a mainstream language, it has received more and more support for environment-based solutions. How is this book to maintain a balance, then? The book's strategy is as follows: If significantly different implementation models exist across C++ compilers, I present a discussion of at least two models. If subsequent implementation models evolved as an attempt to solve perceived problems with the original `cfront` model, as, for example, with support for virtual inheritance, I present a discussion of the historical evolution. Whenever I speak of the traditional implementation model, I mean, of course, Stroustrup's original design as reflected in `cfront` and which has provided a pattern of implementation that can still be seen today in all commercial implementations, even if only as a "reaction against."

## Organization of This Book

Chapter 1, *Object Lessons*, provides background on the object-based and object-oriented programming paradigms supported by C++. It includes a brief tour of the Object Model, illustrating the current prevailing industry implementation without looking too closely at multiple or virtual inheritance. (This is fleshed out in Chapters 3 and 4.)

Chapter 2, *The Semantics of Constructors*, discusses in detail how constructors work. It discusses when constructors are synthesized by

the compiler and what that means in practical terms for your program's performance.

Chapters 3 through 5 contain the primary material of the book. There, the details of the C++ Object Model are discussed. Chapter 3, *The Semantics of Data*, looks at the handling of data members. Chapter 4, *The Semantics of Function*, focuses on the varieties of member functions, with a detailed look at virtual function support. Chapter 5, *Semantics of Construction, Destruction, and Copy*, deals with support of the class model and object lifetime. Program test data is discussed within each of these chapters, where our performance expectations are compared against actual performance as the representations move from an object-based to object-oriented solution.

Chapter 6, *Runtime Semantics*, looks at some of the Object Model behavior at runtime, including the life and death of temporary objects and the support of operators `new` and `delete`.

Chapter 7, *On the Cusp of the Object Model*, focuses on exception handling, template support, and runtime type identification.

## The Intended Audience

This book is primarily a tutorial, although it is aimed at the intermediate C++ programmer rather than the novice. I have attempted to provide sufficient context to make it understandable to anyone who has had some prior exposure to C++—for example, someone who has read my *C++ Primer*—and some experience in C++ programming. The ideal reader, however, has been programming in C++ for a few years and wants to better understand what is actually going on “under the hood.” Portions of the material should be of interest even to the advanced C++ programmer, such as the generation of temporaries and the details of the named return value optimization. At least, this has proved to be so in the various public presentations of this material I have given as it has evolved.

## A Note on Program Examples and Program Execution

The use of program code in this text serves two primary purposes:

1. To provide concrete illustrations of the various aspects of the C++ Object Model under discussion
2. To provide test cases by which to measure the relative cost of various language features

In neither case is the code intended to represent models of production-quality programming. I am not, for example, suggesting that a real 3D graphics library represents a 3D point using a virtual inheritance hierarchy (although one can be found in [POKOR94]).

All the test programs in the text were compiled and executed on an SGI Indigo2 XL (the R4400 MIPS RISC processor) running version 5.2 of SGI's UNIX operating system under both its CC and NCC compilers. CC is cfront Release 3.0.1 (it generates C code, which a C compiler then recompiles into an executable). NCC is version 2.19 of the Edison Design Group's C++ front-end with a code generator supplied by SGI. The times were measured as the average user time reported by the UNIX `timex` command and represent 10 million iterations of the test function or statement block.

While the use of these two compilers on the SGI hardware might strike the reader as somewhat esoteric, I feel doing so serves the book's purposes quite well. Both cfront and now the Edison Design Group's front-end (reportedly characterized by Bjarne as the *son of cfront*) are not platform specific. Rather, they are generic implementations licensed to over 34 computer manufacturers (including Cray, SGI, and Intel) and producers of software environments (including Centerline and Novell, which is the former UNIX Software Laboratories). Performance measurements are intended not to provide a benchmark of current compilation systems but to provide a measure of the relative costs of the various features of the C++ Object Model. Benchmark performance numbers can be found in nearly any "compiler shoot-out" product review in the trade press.

## Acknowledgments

One reason people write books is to set down and share their expertise with others. A second, more selfish reason is to enlarge on and fine tune that expertise. A third is to provide for the public acknowledgment of those who provide the foundation for one's work.

I owe a deep debt of gratitude to many former colleagues at Bell Laboratories without whose encouragement and insight little or nothing of this work could have been accomplished. In particular, Barbara Moo, Andy Koenig, and Bjarne Stroustrup have challenged and supported me throughout the years. Warm appreciation also goes to the Grail gang—Steve Buroff, Martin Carroll, Rob Murray, and Judy Ward—which has been a foundation for many years.

Michael Ball, now at SunPro, generously shared his expertise both through e-mail exchanges and an in-depth review of the text. Doug Schmidt, Cay Horstmann, Greg Comeau, and Steve Clamage also provided

tough, thoughtful reviews of the manuscript that were invaluable in helping me push the manuscript's development forward. Jonathan Shapiro taught me a great deal while we worked together at Bell Laboratories; nuggets of his insight are scattered throughout the text. José Lajoie fielded all too many questions about Standard C++ both with astounding patience and fearful insight.

In addition, I'd like to acknowledge my current foundation here at Walt Disney Feature Animation: Michael Blum, Nhi Casey, Shyh-Chyuan Huang, Scott Dolim, Elena Driskill, Ed Leonard, David Remba, Cary Sandvig, and Dave Tonnesen. Chyuan, Scott, and Elena provided thoughtful readings on various versions of the text. Appreciation also goes to M. J. Turner, Kiran Joshi, Scott Johnston, Marcus Hobbs, and, finally, to the Technology Division management of Dean Schiller and Paul Yanover. They have all helped to make my first year here at Disney sparkle a bit more brightly. A good deal of thanks goes to Dr. Clouis Tondo for detailing errors in the first printing. Thanks also goes to John Potter, Keulin Henny, and Francis Glassborow.

This material has been given at a great many public presentations during the more than two years I have worked on it. These include ACM-sponsored lectures in both Silicon Valley and Los Angeles; two presentations in Tel Aviv sponsored by Sela (with particular thanks to Anna); talks at SIGS Conferences: Object Expo London, Object Expo New York, and C++ World; a tutorial at the 1994 ACM Sigplan Conference on Compiler Construction; at the 1994 IBM-sponsored Cascon Conference; and as part of my C++ Short Course sponsored by UCLA Extension. The resultant feedback has proved of immense help in crafting and revising the material.

Deep thanks also goes to my editor, Debbie Lafferty, who provided both sound counsel and unflagging support and always showed the good sense to laugh at my jokes.

Finally, I'd like to extend my appreciation to Rick Friedman, founder and President of Sigs Publications, publisher of the *C++ Report*, for his support and vision while I was editor of that magazine from mid-1992 through 1995. The *C++ Report* was and remains the best timely source of high-quality technical information on C++. Portions of this text were originally published as columns in the magazine while I was editor.

## References

NOTE: Many of the *C++ Report* articles have been collected on *C++ Gems*, edited by Stanley Lippman, SIGS Books, New York, NY (1996).

[BALL92] Ball, Michael, "Inside Templates," *C++ Report* (September 1992).

[BALL93a] Ball, Michael, "What Are These Things Called Templates," *C++ Report* (February 1993).

[BALL93b] Ball, Michael, "Implementing Class Templates," *C++ Report* (September 1993).

[BOOCH93] Booch, Grady and Michael Vilot, "Simplifying the Booch Components," *C++ Report* (June 1993).

[BORL91] *Borland Languages Open Architecture Handbook*, Borland International Inc., Scotts Valley, CA.

[BOX95] Box, Don, "Building C++ Components Using OLE2," *C++ Report* (March/April 1995).

[BUDD91] Budd, Timothy, *An Introduction to Object-Oriented Programming*, Addison-Wesley Publishing Company, Reading, MA (1991).

[BUDGE92] Budge, Kent G., James S. Peery, and Allen C. Robinson, "High Performance Scientific Computing Using C++," *Usenix C++ Conference Proceedings*, Portland, OR (1992).

[BUDGE94] Budge, Kent G., James S. Peery, Allen C. Robinson, and Michael K. Wong, "Management of Class Temporaries in C++ Translation Systems," *The Journal of C Language Translation* (December 1994).

[CARGILL95] Cargill, Tom, "STL Caveats," *C++ Report* (July/August 1993).

[CARROLL93] Carroll, Martin, "Design of the USL Standard Components," *C++ Report* (June 1993).

[CARROLL95] Carroll, Martin and Margaret A. Ellis, *Designing and Coding Reusable C++*, Addison-Wesley Publishing Company, Reading, MA (1995).

[CHASE94] Chase, David, "Implementation of Exception Handling, Part 1," *The Journal of C Language Translation* (June 1994).

[CLAM93a] Clamage, Stephen D., "Implementing New & Delete," *C++ Report* (May 1993).

[CLAM93b] Clamage, Stephen D., "Beginnings & Endings," *C++ Report* (September 1993).

[ELLIS90] Ellis, Margaret A. and Bjarne Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley Publishing Company, Reading, MA (1990).

[GOLD94] Goldstein, Theodore C. and Alan D. Sloane, "The Object Binary Interface—C++ Objects for Evolvable Shared Class Libraries," *Usenix C++ Conference Proceedings*, Cambridge, MA (1994).

[HAM95] Hamilton, Jennifer, Robert Klarer, Mark Mendell, and Brian Thomson, "Using SOM with C++," *C++ Report* (July/August 1995).

[HORST95] Horstmann, Cay S., "C++ Compiler Shootout," *C++ Report* (July/August 1995).

[KOENIG90a] Koenig, Andrew and Stanley Lippman, "Optimizing Virtual Tables in C++ Release 2.0," *C++ Report* (March 1990).

[KOENIG90b] Koenig, Andrew and Bjarne Stroustrup, "Exception Handling for C++ (Revised)," *Usenix C++ Conference Proceedings* (April 1990).

[KOENIG93] Koenig, Andrew, "Combining C and C++," *C++ Report* (July/August 1993).

[ISO-C++95] C++ *International Standard*, Draft (April 28, 1995).

[LAJOIE94a] Lajoie, Josee, "Exception Handling: Supporting the Runtime Mechanism," *C++ Report* (March/April 1994).

[LAJOIE94b] Lajoie, José, "Exception Handling: Behind the Scenes," *C++ Report* (June 1994).

[LENKOV92] Lenkov, Dmitry, Don Cameron, Paul Faust, and Mickey Mehta, "A Portable Implementation of C++ Exception Handling," *Usenix C++ Conference Proceedings*, Portland, OR (1992).

[LEA93] Lea, Doug, "The GNU C++ Library," *C++ Report* (June 1993).

[LIPP88] Lippman, Stanley and Bjarne Stroustrup, "Pointers to Class Members in C++," *Implementor's Workshop*, *Usenix C++ Conference Proceedings* (October 1988).

[LIPP91a] Lippman, Stanley, "Touring Cfront," *C++ Journal*, Vol. 1, No. 3 (1991).

[LIPP91b] Lippman, Stanley, "Touring Cfront: From Minutiae to Migraine," *C++ Journal*, Vol. 1, No. 4 (1991).

[LIPP91c] Lippman, Stanley, *C++ Primer*, Addison-Wesley Publishing Company, Reading, MA (1991).

[LIPP94a] Lippman, Stanley, "Default Constructor Synthesis," *C++ Report* (January 1994).

[LIPP94b] Lippman, Stanley, "Applying the Copy Constructor, Part 1: Synthesis," *C++ Report* (February 1994).

[LIPP94c] Lippman, Stanley, "Applying the Copy Constructor, Part 2," *C++ Report* (March/April 1994).

[LIPP94d] Lippman, Stanley, "Objects and Datum," *C++ Report* (June 1994).

[METAW94] *MetaWare High C/C++ Language Reference Manual*, Metaware Inc., Santa Cruz, CA (1994).

[MICRO92] Jones, David and Martin J. O'Riordan, *The Microsoft Object Mapping*, Microsoft Corporation, 1992.

[MOWBRAY95] Mowbray, Thomas J. and Ron Zahavi, *The Essential Corba*, John Wiley & Sons, Inc. (1995).

[NACK94] Nackman, Lee R., and John J. Barton *Scientific and Engineering C++*, *An Introduction with Advanced Techniques and Examples*, Addison-Wesley Publishing Company, Reading, MA (1994).

[PALAY92] Palay, Andrew J., "C++ in a Changing Environment," *Usenix C++ Conference Proceedings*, Portland, OR (1992).

[POKOR94] Pokorny, Cornel, *Computer Graphics*, Franklin, Beedle & Associates, Inc. (1994).

[PUGH90] Pugh, William and Grant Weddell, "Two-directional Record Layout for Multiple Inheritance," ACM SIGPLAN '90 Conference, White Plains, New York (1990).

[SCHMIDT94a] Schmidt, Douglas C., "A Domain Analysis of Network Daemon Design Dimensions," *C++ Report* (March/April 1994).

[SCHMIDT94b] Schmidt, Douglas C., "A Case Study of C++ Design Evolution," *C++ Report* (July/August 1994).

[SCHWARZ89] Schwarz, Jerry, "Initializing Static Variables in C++ Libraries," *C++ Report* (February 1989).

[STROUP82] Stroustrup, Bjarne, "Adding Classes to C: An Exercise in Language Evolution," *Software: Practices & Experience*, Vol. 13 (1983).

[STROUP94] Stroustrup, Bjarne, *The Design and Evolution of C++*, Addison-Wesley Publishing Company, Reading, MA (1994).

[SUN94a] *The C++ Application Binary Interface*, SunPro, Sun Microsystems, Inc.

[SUN94b] *The C++ Application Binary Interface Rationale*, SunPro, Sun Microsystems, Inc.

[VELD95] Veldhuizen, Todd, "Using C++ Template Metaprograms," *C++ Report* (May 1995).

[VINOS93] Vinoski, Steve, "Distributed Object Computing with CORBA," *C++ Report* (July/August 1993).

[VINOS94] Vinoski, Steve, "Mapping CORBA IDL into C++," *C++ Report* (September 1994).

[YOUNG95] Young, Douglas, *Object-Oriented Programming with C++ and OSF/Motif*, 2d ed., Prentice-Hall (1995).



# Contents

## 1 Object Lessons 1

Layout Costs for Adding Encapsulation 5

### 1.1 The C++ Object Model 6

A simple object model 6/ A table-driven object model 7/  
The C++ object model/ How the object model effects  
programs

### 1.2 A Keyword Distinction 12

Keywords schmeewords 13/ The politically correct  
struct 16

### 1.3 An Object Distinction 18

The type of a pointer 24/ Adding polymorphism 25

## 2 The Semantics of Constructors 31

### 2.1 Default Constructor Construction 32

Member class object with default constructor 34/ Base  
class with default constructor 37/ Class with a virtual  
function 37/ Class with a virtual base class 38/  
Summary 39

### 2.2 Copy Constructor Construction 40

Default memberwise initialization 41/ Bitwise copy se-  
mantics 43/ Bitwise copy semantics—Not! 45/ Resetting  
the Virtual Table Pointer 45/ Handling the Virtual Base  
Class Subobject 47

### 2.3 Program Transformation Semantics 50

Explicit initialization 50/ Argument initialization 51/  
Return value initialization 53/ Optimization at the user  
level 54/ Optimization at the compiler level 55/  
The copy constructor: to have or to have not? 59/  
Summary 61

### 2.4 Member Initialization List 62

## 3 The Semantics of Data 69

### 3.1 The Binding of a Data Member 72