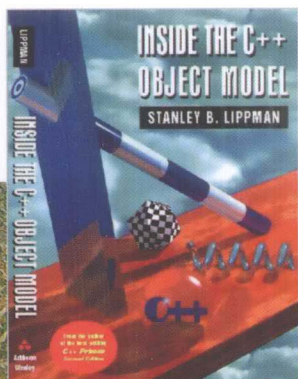


深度探索 C++ 对象模型

[美] Stanley B. Lippman 著
侯捷 译



Inside the C++ Object Model



传世经典书丛
Eternal Classics

深度探索C++对象模型

Inside the C++ Object Model

[美] Stanley B. Lippman 著
侯捷 译

电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

作者 Lippman 参与设计了全世界第一套 C++ 编译程序 cfront, 这本书就是一位伟大的 C++ 编译程序设计者向你阐述他如何处理各种 explicit (明确出现于 C++ 程序代码中) 和 implicit (隐藏于程序代码背后) 的 C++ 语意。

本书专注于 C++ 面向对象程序设计的底层机制, 包括结构式语意、临时性对象的生成、封装、继承, 以及虚拟——虚拟函数和虚拟继承。这本书让你知道: 一旦你能够了解底层实现模型, 你的程序代码将获得多么大的效率。Lippman 澄清了那些关于 C++ 额外负荷与复杂度的各种错误信息和迷思, 但也指出其中某些成本和利益交换确实存在。他阐述了各式各样的实现模型, 指出它们的进化之道及其本质因素。书中涵盖了 C++ 对象模型的语意暗示, 并指出这个模型是如何影响你的程序的。

对于 C++ 底层机制感兴趣的读者, 这必然是一本让你大呼过瘾的绝妙好书。

Authorized translation from the English language edition, entitled *Inside the C++ Object Model*, 1E, 9780201834543 by Stanley B. Lippman, published by Pearson Education, Inc, publishing as Addison-Wesley Professional, Copyright© 1996 by Addison Wesley Longman, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD., and PUBLISHING HOUSE OF ELECTRONICS INDUSTRY Copyright© 2012.

本书简体中文版专有出版权由 Pearson Education 培生教育出版亚洲有限公司授予电子工业出版社。未经出版者预先书面许可, 不得以任何方式复制或抄袭本书的任何部分。

本书简体中文版贴有 Pearson Education 培生教育出版集团激光防伪标签, 无标签者不得销售。

版权贸易合同登记号 图字: 01-2011-4219

图书在版编目 (CIP) 数据

深度探索 C++ 对象模型 / (美) 李普曼 (Lippman, S.B.) 著; 侯捷译. —北京: 电子工业出版社, 2012.1 (传世经典书丛)

书名原文: *Inside the C++ Object Model*

ISBN 978-7-121-14952-8

I. ①深… II. ①李… ②侯… III. ①C 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2011) 第 222573 号

策划编辑: 张春雨

责任编辑: 李云静

印 刷: 北京中新伟业印刷有限公司

装 订: 北京中新伟业印刷有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

开 本: 787×980 1/16 印张: 22.25 字数: 534 千字

印 次: 2012 年 1 月第 1 次印刷

定 价: 69.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zltz@pei.com.cn, 盗版侵权举报请发邮件至 dbqq@pei.com.cn。

服务热线: (010) 88258888。

本立道生

（侯捷 译序）

对于传统的结构化（sequential）语言，我们向来没有太多的疑惑，虽然在函数调用的背后，也有着堆栈建制、参数排列、返回地址、堆栈清除等等幕后机制，但函数调用是那么的自然而明显，好像只是夹带着一个包裹，从程序的某一个地点跳到另一个地点去执行。

但是对于面向对象（Object Oriented）语言，我们的疑惑就多了。究其因，这种语言的编译器为我们（程序员）做了太多的服务：构造函数、析构函数、虚拟函数、继承、多态……有时候它为我们合成出一些额外的函数（或运算符），有时候它又扩张我们所写的函数内容，放进更多的操作。有时候它还会为我们的 objects 加油添醋，放进一些奇妙的东西，使你面对 *sizeof* 的结果大惊失色。

我心里头一直有个疑惑：计算机程序最基础的形式，总是脱离不了一行一行的循序执行模式，为什么 OO（面向对象）语言却能够“自动完成”这么多事情呢？另一个疑惑是，威力强大的 polymorphism（多态），其底层机制究竟如何？

如果不了解编译器对我们所写的 C++代码做了什么手脚, 这些困惑永远解不开。

这本书解决了过去令我百思不解的诸多疑惑。我要向所有已具备 C++多年程序设计经验的同好们大力推荐这本书。

这本书同时也是跃向组件软件 (component-ware) 基本精神的“跳板”。不管你想学习 COM (Component Object Model)、CORBA (Common Object Request Broker Architecture) 或是 SOM (System Object Model), 了解 C++ Object Model, 将使你更清楚软件组件 (components) 设计上的难点与运用之道。不但我自己在学习 COM 的道路上有此强烈的感受, *Essential COM* (《COM 本质论》, 侯捷译, 基峰 1998) 的作者 Don Box 也在他的书中推崇 Lippman 的这一本卓越的书籍。

是的, 这当然不会是一本轻松的书籍。某些章节 (例如 3、4 两章) 可能给你立即的享受——享受于面对底层机制有所体会与掌控的快乐; 某些章节 (例如 5、6、7 三章) 可能带给你短暂的痛苦——痛苦于艰难深涩、难以吞咽的内容。这些快乐与痛苦, 其实就是我翻译此书时的心情写照。无论如何, 我希望通过我的译笔, 把这本难得的好书带到更多人面前, 引领大家见识 C++底层建设的技术之美。

侯捷 2011.10.20 于新竹
jjhou@ccca.nctu.edu.tw

请注意：本书特点，作者 Lippman 在其前言中有很详细的描述，我不再多言。翻译用词与心得，记录在第 0 章（译者的话）之中，对您或有导读之功。

请注意：原文本有大大小小约 80~90 个笔误。有的无伤大雅，有的影响阅读顺畅甚巨（如前后文所用符号不一致、内文与图形所用符号不一致——甚至因而导致图片的文字解释不正确）。我已在第 0 章（译者的话）列出所有我找到的错误。此外，某些场合我还会在错误出现之处再加注，表示原文内容为何。这么做不是画蛇添足，也不为彰显什么。我知道有些读者拿着原文书和中译书对照着看，我把原书错误加注出来，可免读者怀疑是否我打错字或是译错了。另一方面也是为了文责自负……唔……万一 Lippman 是对的而 J.J.Hou 错了呢？！我虽有相当把握，但还是希望明白摊开来让读者检验。

前言

(Stanley B. Lippman)

差不多有 10 年之久，我在贝尔实验室（Bell Laboratories）埋首于 C++ 的实现任务。最初的工作是在 `cfront` 上面（Bjarne Stroustrup 的第一个 C++ 编译器），从 1986 年的 1.1 版到 1991 年 9 月的 3.0 版。然后移转到 `Simplifier`（这是我们内部的命名），也就是 `Foundation` 项目中的 C++ 对象模型部分。在 `Simplifier` 设计期间，我开始酝酿本书。

`Foundation` 项目是什么？在 Bjarne 的领导下，贝尔实验室中的一个小组探索着以 C++ 完成大规模程序设计时的种种问题的解决之道。`Foundation` 项目是我们为了构建大系统而努力定义的一个新的开发模型（我们只使用 C++，并不提供多重语言的解决方案）。这是个令人兴奋的工作，一方面是因为工作本身，一方面是因为工作伙伴：Bjarne、Andy Koenig、Rob Murray、Martin Carroll、Judy Ward、Steve Buroff、Peter Juhl，以及我自己。Barbara Moo 管理我们这一群人（Bjarne 和 Andy 除外）。Barbara Moo 常说管理一个软件团队，就像放牧一群骄傲的猫。

我们把 Foundation 想象成一个核心，在那上面，其他人可以为使用者铺设一层真正的开发环境，把它整修为他们所期望的 UNIX 或 Smalltalk 模型。私底下我们把它称为 Grail（传说中耶稣最后晚餐所用的圣杯），人人都想要，但是从来没人找到过！

Grail 使用一个由 Rob Murray 发展出来并命名为 ALF 的面向对象层次结构，提供一个永久的、以语意为基础的表现法。在 Grail 中，传统编译器被分解为数个各自分离的可执行文件。parser 负责建立程序的 ALF 表现法。其他每一个组件（如 type checking、simplification、code generation）以及工具（如 browser）都在程序的一个 ALF 表现体上操作（并可能加以扩展）。Simplifier 是编译器的一部分，处于 type checking 和 code generation 之间。Simplifier 这个名称是由 Bjarne 所倡议的，它原本是 cfront 的一个阶段（phase）。

在 type checking 和 code generation 之间，Simplifier 做什么事呢？它用来转换内部的程序表现。有三种转换风味是任何对象模型都需要的：

1. 与编译器息息相关的转换（Implementation-dependent transformations）

这是与特定编译器有关的转换。在 ALF 之下，这意味着我们所谓的“tentative” nodes。例如，当 parser 看到这个表达式：

```
fct();
```

它并不知道是否 **(a)** 这是一个函数调用操作，或者 **(b)** 这是 overloaded call operator 在 class object *fct* 上的一种应用。默认情况下，这个式子所代表的是一个函数调用，但是当 **(b)** 的情况出现，Simplifier 就要重写并调换 call subtree。

2. 语言语意转换（Language semantics transformations）

这包括 constructor/destructor 的合成和扩展、memberwise 初始化、对于 memberwise copy 的支持、在程序代码中安插 conversion operators、临时性对象，以及对 constructor/destructor 的调用。

3. 程序代码和对象模型的转换 (Code and object model transformations)

这包括对 `virtual functions`、`virtual base class` 和 `inheritance` 的一般支持、`new` 和 `delete` 运算符、`class objects` 所组成的数组、`local static class instances`、带有非常量表达式 (`nonconstant expression`) 之 `global object` 的静态初始化操作。我对 `Simplifier` 所规划的一个目标是：提供一个对象模型体系，在其中，对象的实现是一个虚拟接口，支持各种对象模型。

最后两种类型的转换构成了本书的基础。这意味着本书是为编译器设计者而写的吗？不是，绝对不是！这本书是由一位编译器设计者针对中高级 C++ 程序员所写的。隐藏在这本书背后的假设是，程序员如果了解 C++ 对象模型，就可以写出比较没有错误倾向而且比较有效率的代码。

什么是 C++ 对象模型

有两个概念可以解释 C++ 对象模型：

1. 语言中直接支持面向对象程序设计的部分。
2. 对于各种支持的底层实现机制。

语言层面的支持，涵盖于我的 *C++ Primer* 一书以及其他许多 C++ 书籍当中。至于第二个概念，则几乎不能够于目前任何读物中发现，只有 [ELLIS90] 和 [STROUP94] 勉强有一些蛛丝马迹。本书主要专注于 C++ 对象模型的第二个概念。本书语言遵循 C++ 委员会于 1995 冬季会议中通过的 `Standard C++` 草案（除了某些细节，这份草案应该能够反映出此语言的最终版本）。

C++ 对象模型的第一个概念是一种“不变量”。例如，C++ `class` 的完整 `virtual functions` 在编译时期就固定下来了，程序员没有办法在执行期动态增加或取代其中的某一个。这使得虚拟调用操作得以快速地派送 (`dispatch`) 结果，付出的成本则是执行期的弹性。

对象模型的底层实现机制，在语言层面上是看不出来的——虽然对象模型的语言本身可以使得某些实现品（编译器）比其他实现品更接近自然。例如，`virtual function calls`，一般而言是通过一个表格（内含 `virtual functions` 地址）的索引而决议得知。一定要使用如此的 `virtual table` 吗？不，编译器可以自由引进其他任何变通做法。如果使用 `virtual table`，那么其布局、存取方法、产生时机以及数百个细节也都必须决定下来，而所有决定也都由每一个实现品（编译器）自行取舍。不过，既然说到这里，我也必须明白告诉你，目前所有编译器对于 `virtual function` 的实现法都是使用各个 `class` 专属的 `virtual table`，大小固定，并且在程序执行前就构造好了。

如果 C++对象模型的底层机制并未标准化，那么你可能会问：何必探讨它呢？主要的理由是，我的经验告诉我，如果一个程序员了解底层实现模型，他就能够写出效率较高的代码，自信心也比较高。一个人不应该用猜的方式，或是等待某大师的宣判，才确定“何时提供一个 `copy constructor` 而何时不需要”。这类问题的解答应该来自于我们自身对对象模型的了解。

写本书的第二个理由是为了消除我们对于 C++语言（及其对面向对象的支持）的各种错误认识。下面一段话节录自我收到的一封信，来信者希望将 C++引进于其程序环境中：

我和一群人工作，他们过去不曾写过（或完全不熟悉）C++和 OO。其中一位工程师从 1985 就开始写 C 了，他非常强烈地认为 C++只对那些 `user-type` 程序才好用，对 `server` 程序却不理想。他说如果要写一个快速而有效率的数据库引擎，应该使用 C 而非 C++。他认为 C++庞大又迟缓。

C++当然并不是天生地庞大又迟缓，但我发现这似乎成为 C 程序员的一个共识。然而，光是这么说并不足以使人信服，何况我又被认为是 C++的“代言人”。本书就是企图极尽可能地将各式各样的 `Object facilities`（如 `inheritance`、`virtual functions`、指向 `class members` 的指针……）所带来的额外负荷说个清楚。

除了我个人回答这封信外，我也把此信转寄给 HP 的 Steve Vinoski；先前我曾与他讨论过 C++ 的效率问题。以下节录自他的回应：

过去数年我听过太多与你的同事类似的看法。许多情况下，这些看法是源于对 C++ 事实真相的缺乏了解。在上周，我和一位朋友闲聊，他在一家 IC 制造厂服务，他说他们不使用 C++，因为“它在你的背后做事情”。我连连追问，于是他说根据他的了解，C++ 调用 *malloc()* 和 *free()* 而不让程序员知道。这当然不是真的。这是一种所谓的迷思与传说，引导出类似于你的同事的看法……

在抽象性和实际性之间找出平衡点，需要知识、经验以及许多思考。C++ 的使用需要付出许多心力，但是我的经验告诉我，这项投资的回报率相当高。

我喜欢把本书想象成是我对那一封读者来信的回答。是的，本书是一个知识陈列库，帮助大家去除围绕在 C++ 四周的迷思与传说。

如果 C++ 对象模型的底层机制会因为实现品（编译器）和时间的变动而不同，我如何能够对于任何特定主题提供一般化的讨论呢？静态初始化（Static initialization）可为此提供一个有趣的例子。

已知一个 class *X* 有着 constructor，如下面所示：

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

而一个 class *X* 的 global object 的声明，如下面所示：

```
X buf;

int main()
{
    // buf 必须在这个时候构造起来
    cin >> setw( 1024 ) >> buf;
    ...
}
```

C++ 对象模型保证, *X constructor* 将在 *main()* 之前便把 *buf* 初始化。然而它并没有说明这是如何办到的。答案是所谓的静态初始化 (*static initialization*), 实际做法则有赖开发环境对此的支持属于哪一层级。

原始的 *cfront* 实现品不单只是假想没有环境支持, 它也假想没有明确的目标平台。唯一能够假想的平台就是 UNIX 及其衍化的一些变体。我们的解决之道也因此只专注在 UNIX 身上: 我们使用 *nm* 命令。CC 命令 (一个 UNIX shell script) 产生出一个可执行文件, 然后我们把 *nm* 施行于其上, 产生出一个新的 .c 文件。然后编译这个新的 .c 文件, 再重新链接出一个可执行文件 (这就是所谓的 *munch solution*)。这种做法是以编译器时间来交换移植性。

接下来是提供一个“平台特定”解决之道: 直接验证并穿越 COFF-based 程序的可执行文件 (此即所谓的 *patch solution*), 不再需要 *nm*、*compile*、*relink*。COFF 是 Common Object File Format 的缩写, 是 System V pre-Release 4 UNIX 系统所发展出来的格式。这两种解决方案都属于程序层面, 也就是说, 针对每一个需要静态初始化的 .c 文件, *cfront* 会产生出一个 *sti* 函数, 执行必要的初始化操作。不论是 *patch solution* 还是 *munch solution*, 都会去寻找以 *sti* 开头的函数, 并且安排它们以一种未被定义的顺序执行 (由安插在 *main()* 之后第一行的一个 library function *_main()* 执行之) (译注: 本书第 6 章对此有详细说明)。

System V COFF-specific C++ 编译器与 *cfront* 的各个版本平行发展。由于瞄准了一个特定平台和特定操作系统, 此编译器因而能够影响链接器特地为它修改: 产生出一个新的 .ini section, 用以收集需要静态初始化的 objects。链接器的这种扩充方

式，提供了所谓的 environment-based solution，那当然更在 program-based solution 层次之上。

至此，任何以 cfront program-based solution 为基础的一般化（泛型）操作将令人迷惑。为什么？因为 C++ 已经成为主流语言，它已经接收了越来越多的 environment-based solutions。本书如何维护其间的平衡呢？我的策略如下：如果在不同的 C++ 编译器上有重大的实现技术差异，我就至少讨论两种做法。但如果 cfront 之后的编译器实现模型只是解决 cfront 原本就已理解的问题，例如对虚拟继承的支持，那么我就阐述历史的演化。当我说到“传统模型”时，我的意思是 Stroustrup 的原始构想（反映在 cfront 身上），它提供一种实现模范，在今天所有的商业化实现品上仍然可见。

本书组织

第 1 章，**关于对象 (Object Lessons)**，提供以对象为基础的观念背景，以及由 C++ 提供的面向对象程序设计范式 (paradigm。译注：关于 paradigm 这个字，请参阅本书第 1 章第 22 页的译注)。本章包括对对象模型的一个概述，说明目前普及的工业产品，但没有对于多重继承和虚拟继承有太靠近的观察（那是第 3 章和第 4 章的重头戏）。

第 2 章，**构造函数语意学 (The Semantics of Constructors)**，详细讨论 constructor 如何工作。本章谈到 constructors 何时被编译器合成，以及给你的程序效率带来什么样的意义。

第 3 章至第 5 章是本书的重要内容。在这里，我详细地讨论了 C++ 对象模型的细节。第 3 章，**Data 语意学 (The Semantics of Data)**，讨论 data members 的处理。第 4 章，**Function 语意学 (The Semantics of Function)**，专注于各式各样的 member functions，并特别详细地讨论如何支持 virtual functions。第 5 章，**构造、析构、拷贝语意学 (Semantics of Construction, Destruction, and Copy)**，讨论如何支持 class 模型，也讨论到 object 的生命期。每一章都有测试程序以及测试数据。我们对效率

的预测，将拿来和实际结果做比较。

第 6 章，**执行期语义学 (Runtime Semantics)**，查看执行期的某些对象模型行为。包括临时性对象的生命及其死亡，以及对 `new` 运算符和 `delete` 运算符的支持。

第 7 章，**在对象模型的尖端 (On the Cusp of the Object Model)**，专注于 `exception handling`、`template support`、`runtime type identification`。

预定的读者

本书可以扮演家庭教师的角色，不过它定位在中级以上的 C++程序员，而非 C++新手。我尝试提供足够的内容，使它能够被任何有点 C++基础（例如读过我的 *C++ Primer* 并有一些实际编程经验）的人接受。理想的读者是，曾经有过数年的 C++编程经验，希望进一步了解“底层做些什么事”的人。书中某些部分甚至对于 C++高手也具有吸引力，如临时性对象的产生，以及 `named return value (NRV)` 最佳化的细节等等。在与本书素材相同的各个公开演讲场合中，我已经证实了这些材料的吸引力。

程序范例及其执行

本书的程序范例主要有两个目的：

1. 为了提供书中所述 C++对象模型各种概念的具体说明。
2. 提供测试，以测量各种语言性质的相对成本。

无论哪一种意图，都只是为了展现对象模型。举例而言，虽然我在书中有大量的举例，但我并非建议一个真实的 3D graphic library 必须以虚拟继承的方式来表现一个 3D 点（不过，你可以在[POKOR94]中发现作者 Pokorny 的确是这么做的）。

书中所有测试程序都在一部 SGI Indigo2xL 上编译执行，使用 SGI 5.2 UNIX 操作系统中的 CC 和 NCC 编译器。CC 是 cfront 3.0.1 版（它会产生出 C 代码，再由一个 C 编译器重新编译为可执行文件）。NCC 是 Edison Design Group 的 C++ front-end 2.19 版，内含一个由 SGI 供应的程序代码产生器。至于时间测量，是采用 UNIX 的 `time` 命令针对 1 000 万次迭代测试所得的平均值。

虽然在 xL 机器上使用这两个编译器，对读者而言可能觉得有些神秘，我却觉得对此书的目的而言，很好。不论是 cfront 或现在的 Edison Design Group's C++ front-end（Bjarne 称其为“cfront 的儿子”），都与平台无关。它们是一种一般化的编译器，被授权给 34 家以上的计算机制造商（其中包括 Gray、SGI、Intel）和软件开发环境厂商（包括 Centerline 和 Novell，后者是原先的 UNIX 软件实验室）。效率的测量并非为了对目前市面上的各家编译系统做评比，而只是为了提供 C++ 对象模型之各种特性的一个相对成本测量。至于商业评比的效率数据，你可以在几乎任何一本计算机杂志的计算机产品检验报告中获得。

致谢

略

参考书目

注意：许多 C++ *Report* 文章已被 C++ *Gems* 收录，并由 Stanley Lippman 编辑完成，参见 *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 Language 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, Josee, "Exception Handling: Behind the Scenes", C++ Report (June 1994)
- [LENKOV92] Lenkov, Dmitry, Don Cameron, Paul Faust, and Michey Mehta, "A Portable Implementation of C++ Exception Handling", Usenix C++ Conference Proceeding, 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, Part1: Synthesis", C++ Report (February 1994)
- [LIPP94c] Lippman, Stanley, "Applying The Copy Constructor, Part2", 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 Crus, CA(1994)
- [MACRO92] 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)