传世经典书丛
Eternal Classics

Essential C++

中文版

[美] Stanley B. Lippman 著
侯捷 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

本书以四个面向来表现C++的本质：procedural（面向过程的）、generic（泛型的）、object-based（基于对象的）、object-oriented（面向对象的）。全书围绕一系列逐渐繁复的程序问题，以及用以解决这些问题的语言特性来组织。循此方式，你将不只学到C++的功能和结构，也可学到它们的设计目的和基本原理。

本书适合那些已经开始从事软件设计，又抽不出太多时间学习新技术的程序员阅读。

版权贸易合同登记号　图字：01-2013-4126

图书在版编目（CIP）数据

Essential C++中文版 /（美）李普曼（Lippman, S. B.）著；侯捷译.——北京：电子工业出版社，2013.8

ISBN 978-7-121-20934-5

Ⅰ. ①E… 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（2013）第150792号

责任编辑：白　涛

印　　刷：三河市双峰印刷装订有限公司

装　　订：三河市双峰印刷装订有限公司

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：18.75　　字　　数：440千字

版　　次：2013年8月第1版

印　　次：2015年1月第3次印刷

定　　价：65.00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zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。www.wzbook.org

服务热线：(010) 88258888。

满汉全席之外（译序/侯捷）

Stanley B. Lippman 所著的 *C++ Primer* 雄踞书坛历久不衰，堪称 C++最佳教科书。但是走过十个年头之后，继 1237 页的 *C++ Primer* 第 3 版，Lippman 又返璞归真地写了这本 276 页的 *Essential C++*。有了满汉全席，为何还眷顾清粥小菜？完成了伟大的巨著，何必回头再写这么一本轻薄短小的初学者用书呢？

所有知道 Lippman 和 *C++ Primer* 的人，脸上都浮现相同的问号。

轻薄短小并不是判断适合初学与否的依据。Lippman 写过 *Inside the C++ Object Model*，280 页小开本，崩掉多少 C++老手的牙。本书之所以号称适合初学者，不在于轻薄短小，在于素材选择与组织安排。

关于 Lippman 重作冯妇的故事，他自己在前言中有详细的介绍。他的转折，他的选择，他的职责，乃至于这本书的纲要和组织，前言中都有详细的交待。这方面我不必再置一词。

身为 *C++ Primer, 3rd Edition* 的译者，以及多本进阶书籍的作者，我必须努力说服自己，才能心甘情愿地将精力与时间用来重复过去的足迹。然而，如果连 Lippman 都愿意为初学者再铺一条红地毯，我也愿意为初学者停留一下我的脚步。

* * * * * * * * * * * * * * * * * * * *

我是一名信息教育者，写译书籍，培训人员，在大学开课……。我真正第一线面对大量学习者。借此机会我要表达的是，所谓“初学者”实在是个过于笼统的名词与分类（呃，谈得上分类吗）。一般所谓“初学者”，多半想象是大一新生程度。其实 C++语言存在各种“初学者”，有 13 岁的，有 31 岁的（当然也有 41 岁的）。只要是第一次接触这个语言，就是这个语言的初学者，他可能才初次接触计算机，可能浸淫 Pascal/C 语言十年之久，也可能已有 Smalltalk/Java 三年经验。有人连计算机基本概念都没有，有人已经是经验丰富的软件工程师。这些人面对 C++，学习速度、教材需求、各人领悟，相同吗？

大不同矣！

每个人都以自己的方式来诠释“初学者”这个字眼，并不经意地反映出自己的足迹。初学者有很多很多种，“初学者”一词却无法反映他们的真实状态。

****

固然，轻薄短小的书籍乍见之下让所有读者心情轻松，但如果舍弃太多应该深入的地方不谈，也难免令人行止失据，进退两难。这本小书可以是你的起点，但绝不能够是你的终站。

作为一本优秀教科书，轻薄短小不是重点，素材选择与组织安排，表达的精准与阅读的顺畅，才是重点。

作为一个好的学习者，背景不是重点，重要的是，你是否具备正确的学习态度。起步固然可从轻松小品开始，但如果碰上大部头巨著就退避三舍逃之夭夭，面对任何技术只求快餐速成，学编程语言却从来不编写程序，那就绝对没有成为高手乃至专家的一天。

有些人的学习，自练一身铜筋铁骨，可以在热带丛林中披荆斩棘，在莽莽草原中追奔逐北。有些人的学习，既未习惯大部头书，也未习惯严谨格调，更未习惯自修勤学，是温室里的一朵花，没有自立自强的本钱。

****

章节的安排，篇幅的份量，索引的保留，习题加解答，以及网上的服务，都使这本小书成为自修妙品、C++专业课程的适当教材。诚挚希望《Essential C++中文版》的完成，帮助更多人获得C++的学习乐趣——噢，是的，OOP（面向对象编程）可以带给你很多乐趣，我不骗你 ☺

侯捷

2012/09/22

敬请注意：

1. 本书与英文版页页对译，从而得以保留原书索引。习题代码下载：www.wzbook.org
2. 本书附加“中英术语对照表”于附录C，并于其中说明中英术语的采用原则。

目录

Contents

满汉全席之外（译序/侯捷） v

前言 Preface xi

结构与组织 xiii

关于源代码 xiii

致谢 xiv

更多读物 xiv

排版约定 xv

第 1 章 C++编程基础 Basic C++ Programming 1

1.1 如何撰写 C++程序 1

1.2 对象的定义与初始化 7

1.3 撰写表达式 10

1.4 条件语句和循环语句 15

1.5 如何运用 Array 和 Vector 22

1.6 指针带来弹性 26

1.7 文件的读写 30

第 2 章 面向过程的编程风格 Procedural Programming 35

2.1 如何编写函数 35

2.2 调用函数 41

2.3 提供默认参数值 50

2.4 使用局部静态对象 53

2.5 声明 inline 函数 55

2.6 提供重载函数 56
2.7 定义并使用模板函数 58
2.8 函数指针带来更大的弹性 60
2.9 设定头文件 63

第 3 章 泛型编程风格 Generic Programming 67
3.1 指针的算术运算 68
3.2 了解 Iterator（泛型指针） 73
3.3 所有容器的共通操作 76
3.4 使用顺序性容器 77
3.5 使用泛型算法 81
3.6 如何设计一个泛型算法 83
3.7 使用 Map 90
3.8 使用 Set 91
3.9 如何使用 Iterator Inserter 93
3.10 使用 iostream Iterator 95

第 4 章 基于对象的编程风格 Object-Based Programming 99
4.1 如何实现一个 Class 100
4.2 什么是构造函数和析构函数 104
4.3 何谓 `mutable`（可变）和 `const`（不变） 109
4.4 什么是 `this` 指针 113
4.5 静态类成员 115
4.6 打造一个 Iterator Class 118
4.7 合作关系必须建立在友谊的基础上 123
4.8 实现一个 copy assignment operator 125
4.9 实现一个 function object 126
4.10 重载 iostream 运算符 128
4.11 指针，指向 Class Member Function 130

第 5 章 面向对象编程风格 Object-Oriented Programming 135
5.1 面向对象编程概念 135
5.2 漫游：面向对象编程思维 138
5.3 不带继承的多态 142

5.4 定义一个抽象基类 145
5.5 定义一个派生类 148
5.6 运用继承体系 155
5.7 基类应该多么抽象 157
5.8 初始化、析构、复制 158
5.9 在派生类中定义一个虚函数 160
5.10 运行时的类型鉴定机制 164

第 6 章 以 template 进行编程 Programming with Templates 167
6.1 被参数化的类型 169
6.2 Class Template 的定义 171
6.3 Template 类型参数的处理 172
6.4 实现一个 Class Template 174
6.5 一个以 Function Template 完成的 Output 运算符 180
6.6 常量表达式与默认参数值 181
6.7 以 Template 参数作为一种设计策略 185
6.8 Member Template Function 187

第 7 章 异常处理 Exception Handling 191
7.1 抛出异常 191
7.2 捕获异常 193
7.3 提炼异常 194
7.4 局部资源管理 198
7.5 标准异常 200

附录 A 习题解答 Exercises Solutions 205
附录 B 泛型算法参考手册 Generic Algorithms Handbook 255
附录 C 中英术语对照 侯捷 271
英文术语的采用原则 271
中英术语对照（按字母顺序排列） 272

索引 Index 277

前言

Preface

天啊，这本书竟是如此轻薄短小。我真想大叫一声“哇欧”！*C++ Primer* 加上索引、扉页、谢词之后，厚达 1237 页，而此书却只有薄薄 276 页。套句拳击术语，这是一部“轻量级”作品。

每个人都会好奇这究竟是怎么回事。的确，这里头有一段故事。

过去数年来，我不断缠着迪斯尼电影动画公司（Disney Feature Animation）的每一个人，请求让我亲身参与一部电影的制作。我缠着导演，甚至 Mickey 本人（如果我可以说出来的话），要求一份管理工作。我会如此疯狂，部分原因是深陷于好莱坞大屏幕那令人神往的无尽魔力而难以自拔。除了计算机科学方面的学位，我还拥有艺术硕士的头衔，而电影工作似乎可以为我带来个人专长的某种整合。我要求管理工作，为的是从制片过程中获取经验，以便提供实际有用的工具。身为一个 C++编译器编写者，我一直都是自己最主要的用户之一。而你知道，当你是自己软件的主力抱怨者时，你就很难再为自己辩护或觉得受到不公平的责难。

《幻想曲 2000》（*Fantasia 2000*）片中有一段火鸟（Firebird）的特效镜头。其计算机特效指导对于我的加盟颇感兴趣。不过，为了掂掂我的斤两，他要求我先写个工具，读入为某段场景所摄的原始数据，再由此产生可嵌入 Houdini 动画套件中的摄影机节点（camera node）。当然，我用 C++把它顺利搞定了。他们爱死它了，我也因此得到了我梦寐以求的工作。

有一次，在制片过程中（在此特别感谢 Jinko 和 Chyuan），我被要求以 Perl 重写那个工具。其他的 TD 并非编程高手，仅仅知道 Perl、Tcl 之类的程序语言。（TD 是电影界的术语，指的是技术导演。我是这部片子的软件 TD，我们还有一位灯光 TD〔你好，Mira〕，一位模型 TD〔你好，Tim〕，以及电影特效动画师〔你好，Mike，Steve，Tonya〕。）而且，喔，天啊，我得赶着点，因为我们想要获得一些观念上的实证，而导演（你好，Paul 和 Gaetan）及特效总监（你好，Dave）正等着结果，准备呈给公司大头目（你好，Peter）。这虽然不是什么紧急要务，可是，你知道的……，唉。

这令我感到些许为难。我自信可以用 C++快速完成，但我不懂 Perl。好吧，我想，我去找本书抱抱佛脚好了——前提是这本书不能太厚，起码此刻不能太厚。而且它最好不要告诉我太多东西，虽然我知道我应该知道每一样东西，不过暂且等等吧。毕竟这只是一场表演：导演们需要一些经过证实的概念，艺术家需要一些东西协助证实其概念，而制片（你好，heck），她需要的是一天 48 小时。此刻我不需要全世界最棒的 Perl 大全，我需要的是一本能妥善引导我前进，使我不致偏离正轨过远的小书。

我找到了 Randal Schwartz 的 *Learning Perl*，它让我立即上手并进展神速，而且颇具阅读趣味。不过，就像其他有趣的计算机书籍一样，它也略去了不少值得一读的内容——尽管在那个时间点，我并不需要了解所有内容，我只需要让我的 Perl 程序乖乖动起来。

我终于在感伤的心境中明白，*C++ Primer* 第三版其实无法扮演人们在初学 C++时的导师角色。它太庞大了。当然，我还是认为它是一本让我骄傲的巨著——特别是由于邀请到 Josée Lajoie 共同完成。但是，对于想立刻学会 C++程序语言的人来说，这本巨著实在过于庞大复杂。这正是本书的由来。

你或许会想，C++又不是 Perl。完全正确！本书也非 *Learning Perl*，它谈的是如何学习 C++。真正的问题在于，谁能够在散尽千页篇幅之后，犹敢自称教导了所有的东西呢？

1. **精细度**。在计算机绘图领域中，精细度指的是影像被描绘出来的鲜明程度。画面左上角那位骑在马背上的匈奴人，需要一张看得清楚眼睛的脸、头发、五点钟方向的影子、衣服……。匈奴人的背后——不，不是那块岩石，老天——唔，相较之下无关紧要。因此我们不会以相同的精细度来描绘这两个影像。同样道理，本书的精细度在相当程度上做了降低。依我看，*C++ Primer* 除了在运算符重载（operator overloading）方面的实例讨论稍嫌不足外，可说极其完备了（我敢这么说是因为 Josée 也有一份功劳）。但尽管如此，*C++ Primer* 还花了 46 页篇幅讨论操作符重载，并附上了范例，而本书却仅以两页带过。

2. **语言核心**。当我还是 *C++ Report* 的编辑时，我常说，杂志编辑有一半工作花在决定哪些题材应该放入，哪些不要。这句话对本书一样成立。本书内容是围绕在编程过程中所发生的一系列问题组织的。我介绍编程语言本身的特性，借此来为不同的问题提供解决之道。书中并未述及任何一个可由多继承或虚继承解决的问题，所以我也就完全没有讨论这两个主题。然而，为了实现 iterator class，我必须引入嵌套类型（nested type）。Class 的类型转换操作符很容易被错用，解释起来也很复杂，所以我不打算在书中提到它。诸如此类。我对题材的选择以及对语言特性的呈现顺序，欢迎大家指教批评。这是我的选择，也是我的职责。

3. **范例的数量**。*C++ Primer* 有数百页代码，巨细糜遗，其中甚至包括一套面向对象的（Object Oriented）文本检索系统，以及十个左右的完整 class。虽然本书也有代码，但数量远不及 *C++ Primer*。为了弥补这项缺憾，我将所有习题解答都置于附录 A。诚如我的编辑 Deborah Lafferty 所言，“如果你想提高教学速度，唾手可得的解答对于学习的强化极有帮助。”

结构与组织

本书由七章和两份附录构成。第 1 章借着撰写一个具有互动性质的小程序，描绘 C++语言预先定义的部分。这一章涵盖了内置的数据类型、语言预定义的运算符（operator）、标准库中的 `vector` 和 `string`、条件语句和循环语句、输入和输出用的 iostream 库。我之所以在本章介绍 `vector` 和 `string` 这两个 class，是因为我想鼓励读者多多利用它们取代语言内置的数组（array）和 C-style 字符串。

第 2 章解释函数的设计与使用，并逐一查看 C++函数的多种不同风貌，包括 inline 函数、重载（*overloaded*）函数、function template，以及函数指针（pointers *to functions*）。

第 3 章涵盖了所谓的 Standard Template Library（STL）：一组容器类（包括 `vector`、`list`、`set`、`map`，等等）、一组作用于容器上的泛型算法（包括 `sort()`、`copy()`、`merge()`，等等）。附录 B 按字典顺序列出了最广为运用的泛型算法，并逐一附上了使用实例。

身为一个 C++程序员，你的主要任务便是提交 class 以及面向对象的 class 层次体系。第 4 章将带领你亲身了解 class 机制的设计与使用过程。在这个过程中，你会看到如何为自身的应用系统建立起专属的数据类型。第 5 章介绍如何扩展 class，使多个相关的 class 形成族系，支持面向对象的 class 层次体系。以我在梦工厂动画电影公司（Dreamworks Animation）担任顾问的经验为例，那时我们设计了一些 class，用来进行四个频道影像合成之类的工作。我们使用了继承和动态绑定（*dynamic binding*）技术，定义影像合成所需的 class 层次体系，而不只是设计八个独立的 class。

第 6 章的重头戏是 class template，那是建立 class 时的一种先行描述，让我们得以将 class 用到的一个（或多个）数据类型或数据值，抽离并参数化。以 vector 为例，可能需要将其元素的类型加以参数化，而 buffer 的设计不仅得将元素类型参数化，还得将其缓冲区容量参数化。本章的行进路线围绕在二分树（binary tree）class template 的实现上。

最后一章，第 7 章，介绍如何使用 C++的异常处理机制（exception handling facility），并示范如何将它融入标准库所定义的异常体系中。附录 A 是本书习题解答。附录 B 提供了关于最广为运用的一些泛型算法的相关讨论与使用实例。

关于源代码

本书的所有程序，以及习题解答中的完整代码，都可从网上获得。你可以在 Addison Wesley 的网站（`http://www.informit.com/store/essential-c-plus-plus-9780201485189`）或博文视点网站（`http://www.broadview.com.cn/20934`）取得。所有程序均在 Visual C++ 5.0 环境中以 Intel C++编

译器测试过，也在 Visual C++ 6.0 环境中以 Microsoft C++编译器测试过。你或许需要稍微修改一下代码才能在自己的系统上编译成功。如果你需要做一些修改并且做了，请将修改结果寄一份给我，我会将它们附上你的大名，附于习题解答代码中。请注意，本书并未显现所有代码。

致谢

在这里，我要特别感谢 C++ *Primer* 第 3 版的共同作者 Josée Lajoie。不仅因为她为本书初稿提供了许多深入见解，更因为她在背后不断地给我鼓舞。我也要特别感谢 Dave Slayton 以他那犀利的绿色铅笔，彻底审阅了文本内容与程序范例。Steve Vinoski 则以同情但坚决的口吻，为本书初稿提供了许多宝贵意见。

特别感谢 Addison-Wesley 编辑团队：全书编辑 Deborah Lafferty 从头到尾支持这个项目；审稿编辑 Besty Hardinger 对本书文字的可读性贡献最大；产品经理 John Fuller 带领我们把一堆文稿化为一本完整的图书。

撰写本书的过程中，我同时还担任独立顾问工作，因此必须兼顾书稿和客户。感谢我的客户对我如此体谅和宽容。我要感谢 Colin Lipworth、Edwin Leonard、Kenneth Meyer，因为你们的耐心与信赖，本书才得以完成。

更多读物

内举不避亲，我要推荐 C++书籍中最好的两本，那便是 Lippman 与 Lajoie 合著的 *C++ Primer*，以及 Stroustrup 的著作 *The C++ Programming Language*。这两本书目前均为第 3 版。我会在本书各主题内提供其他更深入的参考书目。以下是本书的参考书目。（你可以在 *C++ Primer* 和 *The C++ Programming Language* 中找到更广泛的参考文献。）

[LIPPMAN98] Lippman, Stanley and Josée Lajoie, *C++ Primer*, *3rd Editoin*, Addison Wesley Longman, Inc., Reading, MA (1998) ISBN 0-201-82470-1.

[LIPPMAN96a] Lippman, Stanley, *Inside the C++ Object Model*, Addison Wesley Longman, Inc., Reading, MA (1996) ISBN 0-201-83454-5.

[LIPPMAN96b] Lippman, Stanley, Editor, *C++ Gems*, a SIGS Books imprint, Cambridge University Press, Cambridge, England (1996) ISBN 0-13570581-9.

[STROUSTRUP97] Stroustrup, Bjarne, *The C++ Programming Language*, *3rd Editoin*, Addison Wesley Longman, Inc., Reading, MA (1997) ISBN 0-201-88954-4.

[SUTTER99] Sutter, Herb, *Exceptional C++*, Addison Wesley Longman, Inc., Reading, MA (2000) ISBN 0-201-61562-2.

排版约定

本书字体（英文版）为 10.5 pt Palatino。代码和语言关键字为 8.5 pt `lucida`。书中出现的标识符如果后面紧接着 C++的 function call 运算符（也就是一对圆括号`()`），即代表某个函数名称。因此，`foo` 代表程序中的某个 object，`bar()`代表程序中的某个函数。各个 class 的名称以 Palatino 字形呈现。

简体中文版的排版约定是：内文中的一般英文字为 9 pt Times New Roman。代码和语言关键字为 8 pt `Courier New`。各个 class 的名称亦为 8 pt `Courier New`。异常类（exception class）以 8 pt Lucida Sans 呈现。英文长术语（例如 template parameter list，by reference，exception safe）采用 8 pt Arial。运算符名称采用 9 pt Footlight MT Light。译注均以楷体呈现。

1

C++编程基础

Basic C++ Programming

本章，我们将从一个小程序开始，通过它来练习 C++程序语言的基本组成。其中包括：

1. 一些基础数据类型：布尔值（Boolean）、字符（character）、整数（integer）、浮点数（floating point）。
2. 算术运算符、关系运算符以及逻辑运算符，用以操作上述基础数据类型。这些运算符不仅包括一般常见的加法运算符、相等运算符（==）、小于等于（<=）运算符以及赋值（assignment，=）运算符，也包含比较特殊的递增（++）运算符、条件运算符（?:），以及复合赋值（+=等）运算符。
3. 条件分支和循环控制语句，例如 `if` 语句和 `while` 循环，可用来改变程序的控制流程。
4. 一些复合类型，例如指针及数组。指针可以让我们间接参考一个已存在的对象，数组则用来定义一组具有相同数据类型的元素。
5. 一套标准的、通用的抽象化库，例如字符串和向量（`vector`）。

1.1 如何撰写 C++程序

How to Write a C++ Program

此刻，假设我们需要撰写一个简易程序，必须能够将一段信息送至用户的终端（terminal）。信息的内容则是要求用户输入自己的名字。然后程序必须读取用户所输入的名字，将这个名字储存起来，以便后续操作使用。最后，送出一个信息，以指名道姓的方式向用户打招呼。

那么，该从何处着手呢？每个 C++程序都是从一个名为 `main` 的函数开始执行，我们就从这个地方着手吧！`main` 是个由用户自行撰写的函数，其通用形式如下：

```
int main()
{
    // 我们的程序代码置于此处
}
```

int 是 C++程序语言的关键字。所谓关键字（keyword），就是程序语言预先定义的一些具有特殊意义的名称。int 用来表示语言内置的整数数据类型。（下一节我将针对数据类型做更详细的说明。）

函数（function）是一块独立的程序代码序列（code sequence），能够执行一些运算。它包含四个部分：返回值类型（return type）、函数名称、参数列表（parameter list），以及函数体（function body）。下面依次简要介绍每一部分。

函数的返回值通常用来表示运算结果。main() 函数返回整数类型。main() 的返回值用来告诉调用者，这个程序是否正确执行。习惯上，程序执行无误时我们令 main() 返回零。若返回一个非零值，表示程序在执行过程中发生了错误。

函数的名称由程序员选定。函数名最好能够提供某些信息，让我们容易了解函数实际上在做些什么。举例来说，min() 和 sort() 便是极佳的命名。f() 和 g() 就没有那么好了。为什么？因为后两个名称相形之下无法告诉我们函数的实际执行操作。

main 并非是程序语言定义的关键字。但是，执行我们这个 C++程序的编译系统，会假设程序中定义有 main() 函数。如果我们没有定义，程序将无法执行。

函数的参数列表（parameter list）由两个括号括住，置于函数名之后。空的参数列表，如 main()，表示函数不接受任何参数。

参数列表用来表示“函数执行时，调用者可以传给函数的类型列表”。列表之中以逗号隔开各个类型（通常我们会说用户“调用（*call* 或是 *invoke*）”某个函数）。举例来说，如果我们编写 min() 函数，使其返回两数中较小者，那么它的参数列表应该注明两个即将被拿来比较的数值的类型。这样一个用来比较两整数值的 min() 函数，可能会以如下形式定义：

```
int min(int val1, int val2)
{
    // 程序代码置于此处
}
```

函数的主体（body）由大括号（{ }）标出，其中含有“提供此函数之运算”的程序代码。双斜线（//）表示该行内容为注释，也就是程序员对程序代码所做的某些说明。注释的撰写是为了便于阅读者更容易理解程序。编译过程中，注释会被忽略掉。双斜线之后直至行末的所有内容，都会被当作程序注释。

我们的第一件工作就是要将信息送至用户终端。数据的输入与输出，并非 C++程序语言本身定义的一部分（此精神同 C 语言，见 K&R 第 7 章），而是由 C++的一套面向对象的类层次体系（classes hierarchy）提供支持，并作为 C++标准库（standard library）的一员。

所谓类（class），是用户自定义的数据类型（user-defined data type）。class 机制让我们得以将数据类型加入我们的程序中，并有能力识别它们。面向对象的类层次体系（class hierarchy）定义了整个家

族体系的各相关类型，例如终端与文件输入设备、终端与文件输出设备等。（关于类及面向对象的程序设计（object-oriented programming）这两个课题，本书还有许多篇幅会涉及。）

C++事先定义了一些基础数据类型：布尔值（Boolean）、字符（character）、整数（integer）、浮点数（floating point）。虽然它们为我们的编程任务提供了基石，但它们并非程序的重心所在。举个例子，照相机具有一个性质：空间位置。这个位置通常可以用三个浮点数表示。照相机还具备另一个性质：视角方向，同样也可以用三个浮点数表示。通常我们还会用所谓 aspect ratio 来描述照相机窗口的宽/高比，这只需单一浮点数即可表示。

最原始最基本的情况下，照相机可以用七个浮点数来表示，其中六个分别组成了两组 x、y、z 坐标。以这么低级的方式来进行编程，势必会让我们的思考不断地在“照相机抽象性质”和“相应于照相机的七个浮点数”之间反复来回。

class 机制，赋予了我们“增加程序内之类型抽象化层次”的能力。我们可以定义一个 `Point3d` class，用来表示“空间位置”和“视角方向”两个性质。同样的道理，我们可以定义一个 `Camera` class，其中包含两个 `Point3d` 对象和一个浮点数。以这种方式，我们同样使用七个浮点数来表示照相机的性质，不同的是我们的思考不再直接面对七个浮点数，而是转为对 `Camera` class 的操作。

class 的定义，一般来说分为两部分，分别写在不同的文件中。其中之一是所谓的“头文件（header file）”，用来声明该 class 所提供的各种操作行为（operation）。另一个文件，程序代码文件（program text），则包含了这些操作行为的实现内容（implementation）。

欲使用 class，我们必须先在程序中包含其头文件。头文件可以让程序知道 class 的定义。C++标准的“输入/输出库”名为 iostream，其中包含了相关的整套 class，用以支持对终端和文件的输入与输出。我们必须包含 iostream 库的相关头文件，才能够使用它：

```
#include <iostream>
```

我将利用已定义好的 `cout`（读作 see out）对象，将信息写到用户的终端中。output 运算符（<<）可以将数据定向到 `cout`，像下面这样：

```
cout << "Please enter your first name: ";
```

上述这行便是 C++所谓的“语句（statement）”。语句是 C++程序的最小独立单元。就像自然语言中的句子一样。语句以分号作为结束。以上语句将常量字符串（string literal，封装于双引号内）写到了用户的终端。在这之后，用户便会看到如下信息：

```
Please enter your first name:
```

接下来我们要读取用户的输入内容。读取之前，我们必须先定义一个对象，用以储存数据。欲定义一个对象，必须指定其数据类型，再给定其标识符。截至目前，我们已经用过 int 数据类型。但是要用它来储存某人的名字，几乎是不可能的事。更适当的数据类型是标准库中的 string class：

```
string user_name;
```

如此一来我们便定义了一个名为 user_name 的对象，它属于 string class。这样的定义有个特别的名称，称为“声明语句（declaration statement）”。单只写下这行语句还不行，因为我们还必须让程序知道 string class 的定义。因此还必须在程序中包含 string class 的头文件：

```
#include <string>
```

接下来便可利用已定义好的 cin（读作 see in）对象来读取用户在终端上的输入内容。通过 input 运算符（>>）将输入内容定向到具有适当类型的对象身上：

```
cin >> user_name;
```

以上所描述的输出和输入操作，在用户终端上显示如下（输入部分以粗体表示）：

```
Please enter your first name: anna
```

剩下的工作就是打印出向用户打招呼的信息了。我们希望获得下面这样的输出结果：

```
Hello, anna ... and goodbye!
```

当然，这样打招呼稍嫌怠慢。但这不过才第 1 章而已。本书结束之前我会有更具创意的招呼方式。

为了产生上述信息，我们的第一个步骤便是将输出位置（屏幕上的光标）调到下一行起始处。将换行（newline）字符常量写至 cout，便可达到这个目的：

```
cout << '\n';
```

所谓字符常量（character literal）系由一组单引号括住。字符常量分为两类：第一类是可打印字符，例如英文字母（'a'、'A'，等等）、数字、标点符号（';'、'-'，等等）。另一类是不可打印字符，例如换行符（'\n'）或制表符（tab，'\t'）。由于不可打印字符并无直接的表示法（这表示我们无法使用单一而可显示的字符来独立表示），所以必须以两个字符所组成的字符序列来表示。

现在，我们已经将输出位置调整到下一行起始处，接着要产生 Hello 信息：

```
cout << "Hello, ";
```

接下来应该在此处输出用户的名字。这个名字已经储存在 user_name 这个 string 对象中。我们应当如何进行呢？其实就和处理其他数据类型一样，像下面这样即可：

```
cout << user_name;
```

最后我们以道别来结束这段招呼信息（注意，字符串常量内可以同时包含可打印字符和不可打印字符）：

```
cout<< " ... and goodbye!\n";
```

一般而言，所有内置数据类型都可以用同样的方式来输出——也就是说，只需换掉 output 运算符右方的值即可。例如：

```
cout << "3 + 4 = ";
cout << 3 + 4;
cout << '\n';
```

会产生如下输出结果：

```
3 + 4 = 7
```

我们在自己的应用程序中定义了新的 class 时，也应该为每一个 class 提供它们自己的 output 运算符（第 4 章会告诉你如何办到这件事情）。这么一来便可以让那些 class 的用户得以像面对内置类型一样地以相同方式输出对象内容。

如果嫌连续数行的输出语句太烦人，也可以将数段内容连成单一输出语句：

```
cout << '\n'
     << "Hello, "
     << user_name
     << " ... and goodbye!\n";
```

最后，我们以 return 语句清楚地表示 main() 到此结束：

```
return 0;
```

return 是 C++中的关键字。此例中的 0 是紧接于 return 之后的表达式（expression），也就是此函数的返回值。先前我曾说过，main() 返回 0 即表示程序执行成功[1]。

将所有程序片段组合在一起，便是我们的第一个完整的 C++程序：

```
#include <iostream>
#include <string>
using namespace std;    // 此行目前尚未解释……

int main()
{
    string user_name;
    cout << "Please enter your first name: ";
    cin >> user_name;
    cout << '\n'
         << "Hello, "
```

[1] 如果没有在 main() 的末尾写下 return 语句，这一语句会被自动加上。本书各程序范例中，我将不再明确写出 return 语句。

```
            << user_name
            << " ... and goodbye!\n";

    return 0;
}
```

编译并执行后，上述程序代码会产生如下输出结果（输入部分以粗体字表示）：

```
Please enter your first name: anna
Hello, anna ... and goodbye!
```

整个程序中还有一行语句我尚未解释：

```
using namespace std;
```

让我们瞧瞧，如果试着这样解释会不会吓到你（此刻，我建议你深吸一口气）。using 和 namespace 都是 C++中的关键字。std 是标准库所驻之命名空间（namespace）的名称。标准库所提供的任何事物（诸如 string class 以及 cout、cin 这两个 iostream 类对象）都被封装在命名空间 std 内。当然，或许你接下来会问，什么是命名空间?

所谓命名空间（namespace）是一种将库名称封装起来的方法。通过这种方法，可以避免和应用程序发生命名冲突的问题（所谓命名冲突是指在应用程序内两个不同的实体〔entity〕具有相同名称，导致程序无法区分两者。命名冲突发生时，程序必须等到该命名冲突获得解析〔resolve〕之后，才得以继续执行）。命名空间像是在众多名称的可见范围之间竖起的一道道围墙。

若要在程序中使用 string class 以及 cin、cout 这两个 iostream 类对象，我们不仅需要包含<string>及<iostream>头文件，还得让命名空间 std 内的名称曝光。而所谓的 using directive：

```
using namespace std;
```

便是让命名空间中的名称曝光的最简单方法。（[LIPPMAN98]的 8.5 节和[STROUSTRUP97]的 8.2 节有命名空间〔namespace〕的更多相关信息。）

练习 1.1

将先前介绍的 main() 程序依样画葫芦地输入。你可以直接输入程序代码，或是从网络上下载。本书前言已经告诉你如何获得书中范例程序的源程序以及习题解答。试着在你的系统上编译并执行这个程序。

练习 1.2

将 string 头文件注释掉：

```
// #include <string>
```

重新编译这个程序，看看会发生什么事。然后取消对 string 头文件的注释，再将下一行注释：

```
// using namespace std;
```

又会发生什么事情？

练习 1.3

将函数名 main() 改为 my_main()，然后重新编译。有什么结果？

练习 1.4

试着扩充这个程序的内容：(1) 要求用户同时输入名字（first name）和姓氏（last name），(2) 修改输出结果，同时打印姓氏和名字。

1.2 对象的定义与初始化

Defining and Initializing a Data Object

现在，我们的程序引起了用户的注意。让我们出个小题目来考考他。我要显示某数列中的两个数字，然后要求用户回答下一个数字是什么。例如：

```
已知某数列相邻的两个元素值分别为 2 和 3。
试问下一个值是多少?
```

这两个数字事实上是“斐波那契数列（Fibonacci sequence）”中的第三和第四个元素。斐波那契数列的前几个值分别是：1, 1, 2, 3, 5, 8, 13…。斐波那契数列的开头两个数设定为 1，接下来的每个数值都是前两个数值的总和。（第 2 章会示范一个计算斐波那契数列的函数。）

如果用户输入 5，我们就打印出信息，恭喜他答对，并询问他是否愿意试试另一个数列。如果用户输入不正确的值，我们就询问他是否愿意再试一次。

为了提升程序的趣味性，我们将用户答对的次数除以其回答总次数，以此作为评价标准。

这样一来，我们的程序至少需要五个对象：一个 string 对象用来记录用户的名字，三个整数对象分别储存用户回答的数值、用户回答的次数，以及用户答对的次数；此外还需要一个浮点数，记录用户得到的评分。

为了定义对象，我们必须为它命名，并赋予它数据类型。对象名称可以是任何字母、数字、下画线（underscore）的组合。大小写字母是有所区分的，user_name、User_name、uSeR_nAmE、user_Name 所代表的对象各不相同。

对象名称不能以数字开头。举例来说，1_name 是不合法的名称，name_1 则合法。当然，任何命名都不能和程序语言本身的关键字完全一致。例如 delete 是语言关键字，就不可以用于程序内的命名。（这也正是 string class 采用 erase() 而非 delete() 来表示“删去一个字符”的原因。）

每个对象都属于某个特定的数据类型。对象名称如果设计得好，可以让我们直接联想到该对象的属性。数据类型决定了对象所能持有的数值范围，同时也决定了对象应该占用多少内存空间。

前一节中我们已经看过 user_name 的定义。新程序中我们再一次使用相同的定义：

```
#include <string>
string user_name;
```

所谓 class，便是程序员自行定义的数据类型。除此之外，C++还提供了一组内置的数据类型，包括布尔值（Boolean）、整数（integer）、浮点数（floating point）、字符（character）。每一个内置数据类型都有一个相应的关键字，用于指定该类型。例如，为了储存用户输入的值，我们定义一个整数对象：

```
int usr_val;
```

int 是 C++关键字，此处用来指示 usr_val 是个整数对象。用户的“回答次数”以及“总共答对次数”也都是整数，唯一差别是，我们希望为这两个对象设定初值 0。下面这两行可以办到：

```
int num_tries = 0;
int num_right = 0;
```

在单一声明语句中一并定义多个对象，其间以逗号分隔，也可以：

```
int num_tries = 0, num_right = 0;
```

一般来说，将每个对象初始化，是个好主意——即使初值只用来表示该对象尚未具有真正有意义的值。我之所以不为 usr_val 设置初值，是因为其值必须直接根据用户的输入加以设定，然后程序才能使用。

另外还有一种不同的初始化语法，称为“构造函数语法（constructor syntax）”：

```
int num_tries(0);
```

我知道你心中有疑问：为什么需要两种不同的初始化语法呢？为什么直到此刻才提起呢？唔，让我们看看下面这个解释能否回答其中一个问题，甚至同时回答这两个问题。

“用 assignment 运算符（=）进行初始化”这个操作系沿袭自 C 语言。如果对象属于内置类型，或者对象可以单一值加以初始化，这种方式就没有问题。例如以下的 string class：

```
string sequence_name = "Fibonacci";
```

但是如果对象需要多个初值，这种方式就没有办法完成任务了。以标准库中的复数（complex number）类为例，它就需要两个初值，一为实部，一为虚部。于是便引入了用来处理“多值初始化”的构造函数初始化语法（constructor initialization syntax）：