C++ 程序设计语言题解

C++ Solutions
Companion to *The C++ Programming Language*

(美) David Vandevoorde 著
裘宗燕 北京大学 译

机械工业出版社
China Machine Press

HZ BOOKS 华章教育
国外经典教材
Classical Texts From Top Universities

HZ BOOKS
华章教育
北京华章图文信息有限公司

国外经典教材 ❀ Classical Texts From Top Universities

C++程序设计语言题解

C++ Solutions

Companion to The C++ Programming Language

(美) David Vandevoorde 著
裘宗燕 北京大学 译

机械工业出版社
China Machine Press

本书是与Bjarne Stroustrup的《C++程序设计语言》一书配套使用的习题解答，为从《C++程序设计语言》中精选出来的许多练习提供了富有见解的、容易领会的解答，并且附有大量对该书的交叉引用，以便于读者更好地将两本书结合使用。此外，作者对有关练习给出了细致的解释，并为每个选出的练习提供极有价值的提示，以便读者能够找出自己的解答。本书的补充练习提供了对现代软件设计的深入见解，并通过解决一组富有启发性和现实性的练习帮助读者深入理解ANSI/ISO的C++标准。

本书可作为学习C++语言的教学辅导书，也可作为讲授C++程序设计语言的教师的教学参考书。当然，本书同样适于专业程序设计人员使用。

David Vandevoorde: C++ Solutions: Companion to The C++ Programming Language. ISBN 0-201-30965-3.

本书版权登记号：图字：01-2002-4817

图书在版编目（CIP）数据

《C++程序设计语言》题解/（美）范德伍尔杜（Vandevoorde, D.）著；裘宗燕译. - 北京：机械工业出版社，2003.1

（国外经典教材）

书名原文：C++ Solutions：Companion to The C++ Programming Language

ISBN 7-111-11184-2

Ⅰ.C… Ⅱ.①范… ②裘… Ⅲ.C语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2002）第088414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：杨海玲

北京第二外国语学院印刷厂印刷·新华书店北京发行所发行

2003年1月第1版第1次印刷

787mm×1092mm 1/16·15印张

印数：0 001-5 000册

定价：23.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

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，不仅擘划了研究的范畴，还揭橥了学术的源变，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、从业人员较少的现状下，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图文信息有限公司较早意识到“出版要为教育服务”。自1998年始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray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、研究及庋藏。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，有的还专诚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，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步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步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：针对本科生的核心课程，剔抉外版菁华而成“国外经典教材”系列；对影印版的教材，则单独开辟出“经典原版书库”；定位在高级教程和专业参考的“计算机科学丛书”还将保持原来的风格，继续出版新的品种。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

“国外经典教材”是响应教育部提出的使用外版教材的号召，为国内高校的计算机本科教学度身订造的。在广泛地征求并听取丛书的“专家指导委员会”的意见后，我们最终选定了这20多种篇幅内容适度、讲解鞭辟入里的教材，其中的大部分已经被M.I.T.、Stanford、U.C. Berkley、C.M.U.等世界名牌大学采用。丛书不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

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，但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。教材的出版只是我们的后续服务的起点。华章公司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：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995265
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

注意：
- 本书页边标注为英文原书页码，便于读书参考原书的内容。
- 索引中标注英文原书页码，用于查阅。

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

译 者 序

这不是一本寻常的“题解”书。许多常见的题解书中只是简单地罗列原书中一个个习题的答案（也常能看到不那么美妙的“答案”），有时稍微加一点解释。本书则完全不同，作者从《C++程序设计语言》一书的练习中精心挑选出一批具有代表性的题目，用许多篇幅讨论有关情况和背景，分析题目，讨论各种解决方案的优劣。在给出典型解决方案的同时，常常又提出了进一步考虑的问题，给出一些补充性练习。

程序设计的练习原本就应该是这样。一个稍微有点内容的程序题目，通常都可写出许多有各种差异、各具特色，但又都“正确”的程序。如果面对更实际的应用性题目，在分析问题时的不同考虑和决策，会导致千差万别的最终程序，其功能甚至大相径庭，然而又各有各的道理，各以其独有的方式与题目相关联。如果学生给出了这样的解，任何理性的老师都会承认他们确实完成了这个作业题。由此可见，那种针对某书中每个习题简单给出程序的题解书，确实是在代替（或者扼杀）学习者思维，因此不值得提倡。本书则是一个好题解书籍的示例。如果这一示例能对改变国内计算机题解书籍的内在质量起一点帮助作用，那就更值得庆幸了。

当然，技术书籍都有正确使用的问题，作者对此给出了一些值得重视的建议。如果按正确方式使用，David Vandevoorde的这本书一定能在我们学习C++语言及其程序设计的过程中起到更大的作用。总之，本书的目标是启发《C++程序设计语言》一书读者的思维，帮助克服一些难点，是《C++程序设计语言》一书的一个很好补充。

我很高兴能在将本书介绍给广大读者的过程中做一些工作。在此我还要特别感谢机械工业出版社华章公司的杨海玲和温丹丹编辑，感谢她们对本书和《C++程序设计语言》一书的认真工作。这些工作提高了成书的质量，也帮我避免了许多愚蠢错误。

裘宗燕

2002年9月于北京大学

译者简介

裘宗燕，北京大学数学学院信息科学系教授。长期从事计算机软件与理论、程序设计语言和符号计算方面的研究和教学工作。已出版多部著作和译著，包括：《程序设计语言基础》（译著，北京大学出版社，1990），《Mathematica数学软件系统的应用与程序设计》（编著，北京大学出版社，1994），《计算概论（上）》（合著，高等教育出版社，1997），《从问题到程序——程序设计与C语言引论》（编著，北京大学出版社，1999），《程序设计实践》（译著，机械工业出版社，2000），《C++语言的设计和演化》（译著，机械工业出版社，2002），《程序设计语言——概念和结构》（合译，机械工业出版社，2002），《C++程序设计语言（特别版）》（译著，机械工业出版社，2002）等。

致　谢

谢谢各位！

本书的出版应归功于许多人的贡献。

本书起源于Deborah Lafferty（Addison-Wesley的编辑）和Bjarne Stroustrup与我联系，鼓动我写这本书——我感谢他们二人，为他们的初始推动，也为他们在本书的整个撰写过程中所提供的许多建议。Marina Lang和Mike Hendrickson也在本书的设计中起了作用。我还要感谢Mike有关Adobe FrameMaker过程的有益演示。

这本书起源或许可追溯到我第一次借到Bjarne Stroustrup的书，或许可以追溯到他决定设计一个语言，并因此真正提升了我对程序设计的愉悦感觉。因此，我必须感谢那些直接或者间接教授我C++各种要素的人们，特别是Bjarne Stroustrup、Josee Lajoie、Bill Gibbons、John Spicer和Steve Adamczyk。本书中所写的许多内容也是我从各种C++ Usenet论坛的讨论中学到的，这一学习过程在我作为志愿者处理由comp.lang.c++.moderated和comp.std.c++的调解员Matt Austern、Stephen Clamage、Domenic De Vitto、Howard Harkness、Fergus Henderson、James Kanze、Dietmar Kuhl、Robert Martin、John Potter和Herb Sutter送来的工作中一直在继续着。

Eva Wong容忍我长时间地在我们的家中打字和嘟嘟囔囔。我还要感谢我父母和岳父母不断地给我鼓励。

惠普的Michael Beckmann、Sassan Hazeghi和Chris Maitz对于我同时开展这个题目和其他专业工作提供了最大的帮助和支持。

Franklin Luk值得称颂。他耐心地教我写技术性的文字（还教我其他许多东西），我希望他的努力没有白费。

从审阅人那里得到的更正和建议使本书获益良多，他们是：David Francis、Jeffrey Oldham、Srikanth Sankaran、Ed Schiebel、Jay Shain、Bjarne Stroustrup、Clovis Tondo和Chris Van Wyk。他们的贡献怎么评价都不会过分。

David Vandevoorde

贝尔蒙特，加利福尼亚

目　录

第1章 引 论

无论选择什么程序设计语言去编写高效的实际程序，软件开发都不会是简单的事，而且通常总是很复杂的。Bjarne Stroustrup的《C++程序设计语言（特别版）》展示了C++语言如何提供大量的工具，帮助我们分解难点、各个击破，以应对现代软件设计所提出的各种挑战。

解决这些挑战确实需要一定的经验，这转而又要求实践。并不存在任何能“提供实践”的书，但是，Stroustrup提供了许多练习，它们可以用于复习所学的知识，以获取经验。本书为选出的一些练习提供了示例性的答案和讨论。

整体组织结构

本书中选出用于讨论的练习通常具有以下三个共同的特点：

1. 它们阐释了C++中重要的概念，或者阐明了某些常见的使用和方法。
2. 它们的解基本上不依赖于特定的平台。
3. 它们的解写起来相当简洁。

在某些我认为值得去做相关讨论的地方，我偶然也会偏离上述规则。例如，有几个练习所附的解答甚至长达几页。

为了便于查阅，我按照Bjarne Stroustrup的章节组织本书。例如，你将在本书的第15章找到有关《C++程序设计语言》第15章中练习的讨论。

为贯彻本书的组织方式，我不得不设法确定在书中第2章和第3章里做些什么。在《C++程序设计语言》里，这两章带领读者对语言和库做了一次巡游，但它们又都没有提供练习。我的选择是用这两章给出一些简单介绍，介绍C++语言的一些基本概念，以及语言发展所产生的一些结果。原书第23章和第24章也没有练习，在本书中留下了空白的两章。

本书不是一个完整的教程或参考手册。与此相反，书中常常将讨论引回到《C++程序设计语言》的特别版。对特定段落的引证将直接给出章节编号，例如，“C.3.1节”表示引用Stroustrup书中附录C的有关小节。当然，我也常常详细讨论C++语言中一些不那么常用的结构。如果某些常用结构可能产生某些微妙情况的话，有时甚至也讨论这些常用结构。

几乎在每个练习的题目之后都跟着一个短短的提示段落。这些段落指明相关的练习，指明在《C++程序设计语言》里相关材料的位置，并针对该练习所覆盖的基本的或者高级的问题给出一些见解。

大部分练习可以通过许多方法去求解。当不存在某种特别优越的方法时，我常常建议多种可以互相替代的方法。甚至在我确实认为某种特殊解法具有技术优越性时，我也可能完全为了说明而去介绍另一种较为简单的方法。许多问题——包括一些比较简单的问题——都可能引出许多远离主题的试验和讨论，在这种情况下，我有时会用“补充练习”的方式指出这类可以考虑的方向。这是一些新的练习，并未出现在Stroustrup的书中。它们可以用于进一步揭示各种各样的软件开发挑战。请注意，这些练习所涉及的求解过程完全可能超出了本书中给出的解答的范围。

对练习的简短指南

《C++程序设计语言》一书中的练习是希望能成为一种媒介，帮助拓展人们对于语言中各种可能情况的理解。作为一个整体，它们覆盖了C++程序设计语言的知识和求解元素中相当广阔的范围。然而我发现，这里所选出的大部分练习可以归到下面所提出的几个类别中的某一类或某几类：

- *事实和结构*：这些练习可用于检查一个人对语言规范的熟悉程度。下列练习都属于这一类：4.2、4.3、4.7、5.1、5.5、5.6、6.1、6.2、6.5、6.6、6.7、6.9、7.1、7.2、7.14、8.4、8.5、8.7、9.9、10.1、10.4、11.1、11.7、12.1、13.1、13.15、15.2、16.15、20.1。
- *常见惯用法和良好实践*：阐释C++语言常常怎样用于解决各种各样小的程序设计问题。参见练习5.3、5.4、5.7、5.10、5.11、5.12、5.13、6.3、6.10、6.11、6.12、6.13、6.14、6.16、6.17、6.19、6.22、7.3、7.4、7.6、7.7、7.9、7.11、7.16、7.18、7.19、8.1、8.10、9.5、10.2、10.5、10.12、10.19、11.3、11.4、11.8、14.1、14.2、14.9、15.3、16.1、16.2、16.3、16.6、17.3、17.6、18.2、18.5、18.10、18.11、18.14、19.1、19.2、19.3、20.5、20.16、21.1、21.2、21.12、21.18、21.24、22.1、22.2、22.4、25.1、25.5、25.7、25.19。
- *试验*：这些练习所包含的程序或者是用于阐明某些依赖于实现的行为，或者是有关性能的问题。有些练习包含一些小的基准测试，对于获取对各种C++特征和技术的代价的直观理解，这些练习将很有价值。这方面的练习有4.3、4.4、4.5、4.6、5.8、6.8、8.6、8.8、9.1、7.1、18.19、20.15。
- *高级惯用法和设计技术*：讨论一些更特殊的问题。对于这些练习，我通常会试着去强调能够引出一个解答的某种一般思维过程。要将由此得到的思想应用于你的具体情况，还需要做一些并不简单的调整。这样的练习包括7.19、12.10、13.2、13.16、14.10、21.13、22.8。
- *琐碎的和令人讨厌的东西*：每种程序设计语言都有它自己小小的偏狭之处，对这些的滥用将导致晦涩难懂的程序。少数几个练习探索了软件开发中这些比较阴暗的部分。也有一些这类东西虽然无害，但也并非最基本的。这方面的练习有4.6、5.9、6.15、10.15、11.10、11.20、11.21、12.9。

事实上，大部分练习都属于前两个类别，这说明该书强调的还是C++的基本内容。当然，一旦建立了这种基础，更高级的材料也就会变得更容易理解，并能获得更高的工作效率。与此相反，属于*试验*一类的练习通常并不需要熟悉C++的高级特征。

建议

不要强迫自己严格按编号顺序去做这些练习。采取下面方式或许会有所帮助：开始先做属于第一个类别的练习，增强一下你对最基本的东西的理解；而后再去做属于第二类别的练习（*常见惯用法和良好实践*），使自己能较流畅地把握该章所讨论的基本语言元素。与《C++程序设计语言》一样，本书的索引也提供了对有关感兴趣题目的快速而易用的引证。

通常，你不必在完全熟悉了《C++程序设计语言》的一章中的所有材料之后再去试着做练习。当然，通过阅读Stroustrup的书的第2章和第3章（有关“概览”的两章），取得对这种语

言的一个总体认识，则是非常有意义的事情。

特别有趣的是那些长一些的练习。练习15.3是一个流行棋盘游戏的完整的面向对象设计和实现。作为符合标准的基本元素实现的例子，练习18.2是一个算法，练习19.3是一个完整的迭代器的适配器。练习20.5稍短一些，但对此目的而言却很有价值。有关展示标准库威力的例子请见练习6.12，它针对一组数据计算出一些不同的统计量。如果你想寻找一个能磨炼你在更传统的特征方面（语句和表达式）技能的例子，试一下练习6.22。

我相信，要想从本书中获得更多的东西，最好是先不要去看所建议的解答，而是自己去试着做练习。一旦你找到了某个练习的一个解，你可以将自己的解与书中的解比较。在某些情况下，这一比较可能说明哪个解对于该练习是更“自然的解答”。但更多出现的情况是，你可能发现，存在着许多不同的方法去完成同一件工作。如果你在合理的时间内还不能找到一个解，本书中对于该练习的讨论可能提供一些澄清问题的线索。这个讨论常常能引导你考虑出一个与我所提供的解答不同的解，这也是很有价值的实践。

在做这里的许多练习时，最好是实际地去实现有关的解。在描绘一个解的过程中可以学到许多东西，也能使你确信所采用的方式是合理的。当然，实际的实现还需要进一步去关注 3
许多细节问题，良好的软件设计技能也需要包括对于细节的合理考虑方面的磨练。对于试验类别的练习，这一说法就更正确了。在做这些练习时，应当毫不犹豫地去修改这些试验，以使你能进一步认识怎么能使C++程序运行，怎样才能做得更好。

当然，还需要有良好的评价能力，能认清基本机制与具体细节之间的差异。在某些情况下，你可能发现在实现所提出的解时出现了一些令人心烦的情况，原因是所用C++实现并不完全符合语言的ISO规范。第3章将处理这一方面可能遇到的大部分常见问题。

《C++程序设计语言（特别版）》自然是研究所提出的解答时最推崇的信息来源。一旦你熟悉了其中的大部分技术，你也可能希望进一步去钻研更多的技术文献。学术界有许多讨论C++的优秀著作，它们提供了许多超出《C++程序设计语言》讨论范围的材料。我想强调的是下面几本：

Effective C++: 50 Specific Ways to Improve Your Programs and Designs (Second Edition). Scott Meyers. Reading, MA: Addison Wesley Longman, 1997.

More Effective C++: 35 New Ways to Improve Your Programs and Designs. Scott Meyers. Reading, MA: Addison Wesley Longman, 1996.

The Design and Evolution of C++. Bjarne Stroustrup. Reading, MA: Addison Wesley Longman, 1994.（《C++语言的设计和演化》，裘宗燕译，机械工业出版社，2002）

前两本书描述了许多实践性的规则和技术，它们可以导致更强健的，常常也是更高效的C++程序。书中介绍了一些具有普遍意义的设计原则和特殊编码技术。

第三本书描述了直至1993年底为止的C++语言的演化过程。我已经深刻地认识到，对这一演化过程的理解，对于我们确定应当怎样最好地使用C++的丰富结构集合极有帮助。它也能帮助我们处理好C++的某些微妙特征，使之更直观、更容易掌握。

最后我还想指出，C++社团非常庞大。与朋友和同事开展有关技术、语义和设计原则方面的讨论是极有价值的。此外，现在有许多有关C++程序设计的会议和电子论坛。特别是，我已经发现Usenet论坛comp.lang.c++.moderated和comp.std.c++中对各种各样的C++问题都有极富见解的讨论。 4

第2章 C++ 概 念

本书无意作为一本C++程序设计语言的独立教材。《C++程序设计语言》是满足这一需要的最佳资源。当然，对一些基本概念做个总结，在术语方面达成一种共识也是很有用的、符合习惯的。

你可以安全地跳过这一章，在需要快速复习某些C++基本问题时再转回来。如果需要更完整的深入的信息，那么请参考Stroustrup的书。

语言和实现

程序设计语言是一个抽象的事物，它是一组描述程序行为的规则和约定。对这一抽象事物的具体化就是一个*语言实现*——一个设备，实际上常常是一个程序，它能将一个程序描述转变为（翻译为）一个实际的行为。该转变过程将始终强制性地要求由该程序设计语言所设定的那些规则。

对于C++，在绝大多数情况下，这种实现是一个*编译器*，编译器是一个能够将文本（*源代码*）翻译为*可执行程序*的程序。一旦翻译完成之后，得到的程序就可以在某个*目标平台*（计算机）上重复执行了。

本书假定你已经在一定程度上了解这些概念。某些实践性考虑还将在练习4.1讨论。

词法单词

C++源代码通常被划分为多个文件（9.1节）。这种源文件经过预处理之后，注释都被剔除，宏都被替换掉了。这样得到的结果通常被称为*编译单位*。一个*编译单位*是一个*词法单词*的序列，词法单词包括：单词、数字和各种符号。它们都是C++语言里具有独立意义的最小实体。例如，下面这一小段C++代码


```
int const hundred = 100 ;
```

由六个词法单词组成："int"、"const"、"hundred"、"="、"100" 和 ";"。*标识符*是一类词法单词，其"拼写形式"可以由程序员自由选择。它们是字母、数字和下划线（_）组成的任意序列，但是不能以数字开头。请注意，字母的大小写是有意义的。某些序列虽然符合条件，但是不能作为标识符：

- *关键字*（A.2节）和某些符号的*替代表示形式*："int" 和 "const" 是关键词的例子。"and" 是符号 "&&"（*逻辑与操作*）的替代表示形式的例子。C++程序设计语言中有63个关键词和11个替代表示形式。
- *保留的名字*：为使具体实现能引进自己的名字而保留一些名字有时是方便的。包含连续的两个下划线的名字（如re__served），以及以下划线开头后面紧跟着一个大写字母的名字（如_Reserved）是为此而保留的，因此不应该作为用户定义的标识符。

词法单词通常用*空白字符*分隔，但这些空白字符本身并无意义。在许多情况下，从一个词法

单词到下一个词法单词的翻译并不要求有这种空白的存在，也不限制在词法单词之间的空白字符的个数。例如，上面那个声明也可以以下面两种形式出现：

```
int const hundred=100;
int
   const
      hundred
         =
    100
```

但下面形式就不会工作了：

```
intconst hundred = 100;
```

没有空白字符分隔，intconst将被看做是一个标识符——假定是用户自己定义的——而不是连续的两个关键字。

空白字符的某些使用可能损害源代码的清晰性，我们的第三个例子说明了这类情况。但无论如何，个人的爱好可能不同，这些形成了源代码的不同风格，本书并不想在这个方面去强调某些特定的规则。许多项目确实要求遵守一些C++源代码的格式规则，以便使源文件更容易理解（另见4.9.3节和练习6.23）。

名字、声明和作用域

C++程序里的许多实体都需要通过*名字*去引用。引进或者重新引进名字的描述形式通常称为*声明*。如果在某编译单位的一部分里面，可以通过某个声明所引进的简单名字去引用这个声明，该部分就称为这个名字的*作用域*。例如： 6

```
int a = 3;
int b = 2; // <- 全局b的作用域由此开始

void f() {
   int b = 0; // <- 全局b的作用域在此停止
   ++a;       // 局部b的作用域开始
   ++b; // 引用局部的b
} // <- 局部b的作用域结束，全局b的作用域恢复
```

正如本例所示，两个不同的声明可以关联于同一名字。在出现这种情况时，“最内层”的名字将屏蔽外层名字。在某些情况下，被屏蔽的名字以及并非位于活动作用域里的名字可以通过通常称为*限定名*的方式访问，这种名字包含一个作用域解析运算符“::”。例如：

```
int a;

namespace N {
   int a, b;
   void f() {
      ::a = 1; //（被屏蔽的）全局a的限定名
      a = 10; // 非限定名访问内层a
    }
}

void g() {
   a = 10; // 全局的a
   N::a = 7; // 名字空间N作用域中的a（限定名）
}
```

位于作用域解析运算符左边的限定符必须或者是一个名字空间的名字（8.2节），或者是一个类的名字（10.2节）。这也意味着，局部于某个函数的名字不能通过限定名的方式访问。

对象、类型、引用和函数

对象是那种占据着一块连续存储区域（计算机内存）的东西。然而，随机的一组连续信息位通常并不被看做是对象。相反，一个对象总有一个与其物理的位组相关联的*类型*。类型是一组属性和可应用操作，这些属性和可应用操作确定了组成一个对象的位组应该如何解释。例如，在许多系统里，定义

7

```
char c = 1;
```

和

```
bool b = true;
```

的结果将得到同样的物理的二进制值的集合。然而由于它们的类型不同，++c和++b很可能具有不同的物理值。

C++有一组内部的*基本类型*，bool、char、unsigned int和double是其中的一些例子（4.2节、4.3节、4.4节、4.5节）。其他的类型被称为*复合类型*，可以通过许多不同方式建立。这些复合类型可以归为如下几个类别：

- 枚举（4.8节）
- 类（包括结构，5.7节，第10章）
- 联合（10.4.12节）
- 数组（5.2节）
- 指针（5.1节，5.3节）
- 引用（5.5节）
- 函数（第7章）

虽然函数和引用也具有类型，它们却不是C++语言的意义下的*对象*。引用不过是对象的“别名”——因此，在运行时它们可能并不占用任何存储空间，在某些时候，这些别名可能被C++编译器替换掉。在典型情况下，函数确实需要为它们的表示占用运行时存储空间，但是C++和其他大部分程序设计语言一样，不允许程序员去使用函数所占据的空间。这就使我们无法去写（例如）能自我修改的代码。

左值和右值表达式

对象的存在是因为显式地定义了它们，或者是分配了它们（例如，通过使用new运算符）。如果一个表达式引用这样的对象，这个表达式通常被称为是一个*左值表达式*，否则称为*右值表达式*。例如：

```
int a;
int *b = &a;
a = 2; // a是左值，2是右值
*b = a; // *b和a都是左值
b = new int; // new int是右值，b是左值
int const &ri = *new int; // *new int是左值
a = ri; // ri是左值
```

```
int f();
*b = f(); // f()是右值
int& g();
a = g(); // g()是左值
```

8

术语*左值*（lvalue）中的“*左*”（1）是从C语言的前身（例如BCPL，第1章）那里继承来的，在那些语言里，左值只能出现在赋值的左边，而右值则不能出现在赋值左边。在现代的C以及在C++里，左值却未必出现在赋值左边，因为被引用的对象可以是const。这种左值也被称为*不可修改的左值*。C++还提供了修改右值的方式，但极少需要去用它。

算术运算符（+、-、%、*等）的结果和函数调用的结果通常都是右值，除非它们返回引用。

初始化和赋值

初始化（4.9.5节）是为一个对象建立第一个值的过程，而另一方面，*赋值*（2.3.1节）是一个操作，通常用于改变一个对象的值，其以前的值已经存在了。赋值用词法单词“=”表示，有时初始化也用这一符号表示：

9

```
int a = 12; // 初始化（定义）
a = 13; // 赋值
```

简而言之，当“=”符号出现在声明里的时候，它就表示初始化；否则就表示赋值。

不用“=”符号也可以描述初始化。例如：

```
int a(12); // 与上面一样
```

还有，函数参数的传递也是通过初始化机制。例如：

```
void f(X x) { /* ... */ }

void g() {
   X a; // 假定X是某个类型
   f(a); // 参数x通过初始化取得a的值
}
```

对于类，你可以通过提供构造函数的方式定义自己选择的初始化过程（10.2.3节，10.4节），可以通过提供operator= 的方式定义自己的赋值运算（10.4.4.1节）。请记住，初始化时绝不会调用你的operator=，即使某个初始化是用“=”符号表示的。

声明的语法形式

出现在声明中的大部分常用语法是比较直观的，但最一般的语法规则则常常令人吃惊。下面的讨论总结了第5章、第6章、第7章和第10章中说明的基础知识。

变量和常量

一个声明由以下四部分组成（4.9.1节）：

1. 一组（可选的）描述符。
2. 一个基类型。
3. 一个声明符。

4. 一个可选的初始式。

描述符或者是对应于与被声明实体相关联的一些属性，例如extern（9.2节）、virtual（12.2.6节）和explicit（11.7.1节）；或者是对应于与被声明实体的类型相关联的一些属性（例如const，5.4节）。有趣的是，对于描述符出现的顺序只有很少的限制，而且事实上，它们有时甚至会出现在基类型的后面。例如，下面所有声明合法，而且是等价的：

```
extern int const c;
const int extern c;
const extern int c;
int extern const c;
```

几乎所有程序员都愿意将与类型相关的描述符（*类型描述符*）放得靠近基类型，而将其他描述符放在基类型和类型描述符的前面。上面声明通常被排列为下面两种形式之一：

```
extern const int c; // const是类型描述符
extern int const c; // extern不是
```

前一种排列形式最为流行，但由于练习5.1中将要解释的原因，有时第二种形式更合适。

基类型可以是一个基本类型（如float），也可以是一个enum、class（或struct）或者union的名字。如果这一基类型不是基本类型，其名字可以是*限定的*（见前面）或者*加修饰的*。所谓加修饰的意思是，在一个（可能加了限定的）名字之前可以有下列关键词中的某一个：enum、struct、class和union。例如：

```
namespace N {
   enum E { two = 2, seven = 7 };
   E a; // 名字E无限定也无修饰
   enum E b; // 加修饰但无限定的名字E
}
enum N::E c; // 加修饰且限定的名字E
```

一个声明的*声明符*部分包括被声明的名字，可能还环绕着某些可选的*声明运算符*，它们用于修改与该名字关联的基类型。这些运算符用于创建引用、指针、数组、函数和这些东西的组合。让我们先考虑指针的情况，指针声明时使用一个前缀的星号“*”。指向整数的指针p可以声明如下：

10
```
int *p;
```

上面我们说过如何将关键字const关联于一个基类型。例如，

```
int const *p;
const int *q;
```

意味着p和q是指向整数的指针，但是却不能通过它们去修改它们所指向的整数。有时我们可能希望表达指针本身不能被修改。做这件事情的方式就是将关键字const放在声明运算符“*”之后：

```
int i;
int *const cp = &i;
int const *const cpc = &i;
```

后一声明所描述的指针cpc本身不能修改，被它所指的东西也不能通过cpc修改：

```
*cp = 3; // 可以：不是指向const基类型的指针
int j;
```