引言

致读者

“当实际地形与地图不符时，相信实际地形。”

——瑞士军队谚语

本章汇集了多种信息，目的是使你对本书其余部分的内容有初步了解。你可以略过本章，直接阅读后面你感兴趣的部分。对教师来说，可以发现很多直接有用的内容。如果没有一个好的老师指导你学习本书，请不要试图阅读并理解本章的所有内容，只要阅读“本书结构”一节和“讲授和学习本书的方法”一节的第一部分即可。当你已经能自如编写和执行小程序时，可能需要回过头来重读本章。

本书结构
一般方法
简单练习、习题等
进阶学习
讲授和学习本书的方法
本书内容顺序的安排
程序设计和程序设计语言
可移植性
程序设计和计算机科学
创造性和问题求解
反馈方法
参考文献
作者简介

本书结构

本书由四个部分和若干个附录组成：

- 第一部分：基础知识，介绍了程序设计的基本概念和技术，以及开始编写代码需要了解的一些 C++ 语言和库的知识。这部分包括类型系统、算术运算、控制结构、错误处理，以及函数和用户自定义类型的设计、实现和使用等内容。
- 第二部分：输入和输出，介绍了如何从键盘和文件获取数值和文本数据，以及如何生成相应的输出到屏幕和文件。然后介绍了如何以图形化方式表示数值数据、文本和几何图形，以及如何从图形用户界面（graphical user interface, GUI）获取输入数据。
- 第三部分：数据结构和算法，重点是 C++ 标准库中的容器和算法框架（标准模板库，standard template library, STL）。展示了容器（如向量、列表和映射）是如何（用指针、数组、动态内存、异常和模板）实现的以及如何使用它们。还展示了标准库算法（如排序、查找和内积）如何设计及使用。
- 第四部分：拓宽眼界，通过 C++ 思想和历史的讨论，一些实例（如矩阵运算、文本处理、测试以及嵌入式系统程序设计），以及 C 语言的简单描述，为我们呈现了程序设计的一个全景。
- 第五部分：附录，提供了一些不适合作为教学展示但很有用的内容，如 C++ 语言和标准库的概要介绍，以及集成开发环境（integrated development environment, IDE）和图形用户界面（GUI）的入门简介等。

不过现实世界中的程序设计并不能真正分为完全独立的四个部分。因此，这种划分仅仅是对本书内容的一种粗略分类。我们认为这是一种有用的分类方法（这是显然的，否则我们不会采用它），但现实情况往往与这种简洁的分类法相悖。例如，我们很快就会用到输入操作，但对 C++ 标准输入 / 输出流（input/output stream，I/O 流）的完整介绍却出现在本书比较靠后的部分。书中有些地方在提出某个概念时需要先介绍另外一些内容，而这与全书的布局不符，对此，我们会在此处简明介绍这些内容，以便更好地提出概念，而不是仅仅指出这些内容的完整介绍在书中什么地方。刻板的分类法更适合于手册而不是教材。

本书内容的顺序是由程序设计技术决定的，而不是程序设计语言特性，参见“讲授和学习本书的方法”。附录 A 是按语言特性组织的。

在你第一次阅读时，你可能还没有发现哪些信息是至关重要的，为了方便复习，也为了帮助你不遗漏关键点，我们在页边空白处放置了三种“提醒标记”：

- 圆形：概念和技术（本段就是一个例子）
- 三角形：建议
- 正方形：警告

一般方法

在本书中，人称都是直接的，简单清楚，不像很多科技论文中那种惯用的“专业”的婉转称呼方式。当用到“你”时，我们的意思就是“你——读者”；而“我们”指“我们——作者和教师”，或者指读者和我们一起在讨论某个问题，就好像我们在一个教室中一样。

本书的内容组织适合从头到尾一章一章地阅读，当然，你也常常要回过头来对某些内容读上第二遍、第三遍。实际上，这是一种明智的方法，因为当遇到还看不出什么门道的地方时，你通常会快速掠过。对于这种情况，你最终还是会再次回到这个地方。然而，这么做要有个度，因为尽管有索引和交叉引用，对本书其他部分，你随便翻开一页，就开始学习并希望成功，这几乎是不可能的。本书每一节、每一章的内容安排，都假定你已经理解了之前的内容。

本书的每一章都是一个合理的独立单元，这意味着应将其一口气读完（当然这只是从理论上讲，实际上由于学生紧密的学习计划，不总是可行的）。这是将内容划分为章的主要标准。其他标准包括：从简单练习和习题的角度，一章是一个合适的单元；每一章提出一些特定的概念、思想或技术。这种标准的多样性使得少数章过长，所以不要教条地遵循“一口气读完”的准则。特别是当你已经考虑了思考题，做了简单练习和一些习题时，你通常会发现需要回过头去重读一些小节和几天前读过的内容。我们按主题将章组合成“部分”，例如，第二部分都是关于输入 / 输出的内容。每一部分都可以作为一个很好的完整的复习单元。

“它回答了我想到的所有问题”是对一本教材常见的称赞，这对细节技术问题是很理想化的，早期的读者发现本书有这样的特性。但是，这不是全部的理想，我们希望提出更多初学者可能想不到的问题。我们的目标是，回答那些你在编写供他人使用的高质量软件时需要考虑的问题。学习回答好的（通常也是困难的）问题是学习如何像一个程序员那样思考所必需的。只回答那些简单的、浅显的问题会使你感觉良好，但无助于你成长为一名程序员。

我们努力尊重你的聪明才智，珍惜你的时间。在本书中，我们以专业性而不是精明伶俐为目标，我们宁可有节制地表达一个观点而不大肆渲染它。我们尽力不夸大一种程序设计技术或一种语言特性的重要性，但请不要因此低估“这通常是有用的”这种简单陈述的重要程度。如果我们平静地强调某些内容是重要的，我们的意思是，你如果不掌握它，或早或晚都会因此而浪费时间。我们喜欢幽默，但在本书中使用很谨慎。经验表明，人们对什么是幽默的看法大相径庭，不恰当地使用幽默会把人弄糊涂。

我们不会谎称本书中的思想和工具是完美的。实际上没有任何一种工具、库、语言或者技术能够解决程序员所面临的所有难题，至多能帮助开发和表达你的问题求解方案而已。我们尽量避免“善意的谎言”，也就是说，对于那些清晰易理解，但在实际程序设计和问题求解时容易弄错的内容，对其介绍避免过度简单化。另外，本书不是一本参考手册；如果需要 C++ 详细完整的描述，请参考 Bjame Stroustrup 的 *The C++ Programming Language, Fourth Edition* 一书（Addison-Wesley 出版社，2013 年）和 ISO 的 C++ 标准。

简单练习、习题等

程序设计不仅是一种脑力活动，实际动手编写程序是掌握程序设计技巧必不可少的一个环节。本书提供以下两个层次的程序设计练习：

- 简单练习：简单练习是一种非常简单的习题，其目的是帮助学生掌握一些机械的实际程序设计技巧。一个简单练习通常由对单个程序的一系列修改组成。你应该完成所有的简单练习。完成简单练习不需要很强的理解能力、很聪明或者很有创造性。简单练习是本书的基本组成部分，如果你没有完成简单练习，就不能说完成了本书的学习。
- 习题：有些习题比较简单而有些则很难，但大多数习题都是想给学生留下一定的创造和想象的空间。如果时间紧张，你可以做少量习题。但至少应该弄清楚哪些内容对你来说比较困难，在此基础上应该再多做一些，这才是学习成功之道。我们希望本书的习题都是学生能够做出来的，而不是需要超乎常人的智力才能解答的复杂难题。但是，我们还是期望本书习题能给你足够多的挑战，能耗尽最优秀学生的所有时间。我们不期待你能完成所有习题，但请尽情尝试。

另外，我建议每个学生都能参与到一个小的项目中去（如果时间允许，能参与更多项目当然就更好了）。一个项目就是要编写一个完整的有用的程序。在理想情况下，一个项目由一个多人小组（比如三个人）共同完成，最好在学习第三部分的同时花大概一个月时间来完成整个项目。大多数人会发现做项目非常有趣，并在这个过程中学会如何把很多事情组织在一起。

一些人喜欢在读完一章之前就把书扔到一边，开始尝试做一些实例程序；另一些人则喜欢把一章读完后，再开始编码。为了帮助前一种读者，我们在正文的段落之间放置了一些有“试一试”标识的文字，给出了对于程序设计实践的一些简单建议。“试一试”本质上来说就是一个简单练习，而且只着眼于前面刚刚介绍的主题。如果你略去一个“试一试”而没有去尝试它（也许因为你的手边没有计算机，或者你过于沉浸在正文的内容中），那么最好在做这一章的简单练习时做一下这个题目。“试一试”要么是该章简单练习的补充，要么干脆就是其中一部分。

在每章末尾你都会看到一些思考题，我们设置这些思考题是想为你指出这一章中的重点内容。学习思考题的方法是把它们作为习题的补充：习题关注程序设计的实践层面，而思考题则试图帮你强化思想和概念。这么看，思考题有点像面试题。

每章最后都有“术语”一节，给出本章中提出的程序设计或 C++ 方面的基础词汇表。如果你希望理解别人关于程序设计的陈述，或者想明确表达出自己的思想，那么应该首先弄清术语表中每个术语的含义。

重复是学习的有效手段，我们希望每个重要的知识点都在书中至少出现两次，并通过习题再次加强。

进阶学习

当你完成本书的学习时，是否能成为一名程序设计和 C++ 方面的专家呢？答案当然是否定的！如果做得好的话，程序设计会是一门建立在多种专业技能上的精妙的、深刻的、需要高度技巧的艺术。你不能奢望花四个月时间就成为一名程序设计专家，就像你不能奢望花四个月、半年或一年时间就成为一名生物学专家、数学家、自然语言（如中文、英文或丹麦文）方面的专家或者小提琴演奏家一样。如果你认真地学完了这本书，你可以期待，也应该期待的是：你已经在程序设计领域有了一个很好的开始，已经可以写相对简单有用的程序，能读更复杂的程序，而且已经为进一步的学习打下了良好的理论和实践基础。

学习完这门入门课程后，最好的进一步学习的方法是开发一个真正能被别人使用的程序。在完成这个项目之后，或者同时（同时可能更好），学习一本专业水平的教材（如 Stroustrup 的 *The C++ Programming Language*），学习一本与你做的项目相关的更专业的书（例如，你如果在做 GUI 相关项目的话，可选择关于 Qt 的书；如果在做分布式程序的话，可选择关于 ACE 的书），或者学

习一本专注于 C++ 某个特定方面的书（如 Koenig 和 Moo 的 *Accelerated C++*，Sutter 的 *Exceptional C++* 或 Gamma 等人的 *Design Patterns*）。完整的参考书目，参见本章的参考文献一节或本书最后的参考文献。

最后，你应该学习另一门程序设计语言。我们认为，如果只懂一门语言，你是不可能成为软件领域的专家的（即使你并不想做一名程序员）。

讲授和学习本书的方法

我们是如何帮助你学习的？又是如何安排教学进程的？我们的做法是，尽力为你提供编写高效实用程序所需的最基本的概念、技术和工具，包括：

- 程序组织；
- 调试和测试；
- 类设计；
- 计算；
- 函数和算法设计；
- 绘图方法（仅介绍二维图形）；
- 图形用户界面（GUI）；
- 文本处理；
- 正则表达式匹配；
- 文件和流输入输出（I/O）；
- 内存管理；
- 科学 / 数值 / 工程计算；
- 设计和程序设计思想；
- C++ 标准库；
- 软件开发策略；
- C 语言程序设计技术。

认真完成这些内容的学习，我们会学到如下程序设计技术：过程式程序设计（C 语言程序设计风格）、数据抽象、面向对象程序设计和泛型程序设计。本书的主题是程序设计，在代码中表达想法需要的思想技术和工具。C++ 语言是我们的主要工具，因此我们比较详细地描述了很多 C++ 语言的特性。但请记住，C++ 只是一种工具，而不是本书的主题。本书主题是“用 C++ 语言进行程序设计”而不是“C++ 和一点程序设计理论”。

我们介绍的每个主题都至少有两个目的：提出一种技术、概念或原理，介绍一种实用的语言特性或库特性。例如，我们用一个二维图形绘制系统的接口展示如何使用类和继承。这使我们节省了篇幅（也节省了你的时间），并且还强调了程序设计不只是简单地将代码拼装起来以尽快地得到一个结果。C++ 标准库是这种“双重作用”例子的主要来源，其中很多主题甚至具有三重作用。例如，我们会介绍标准库中的向量类 vector，用它来展示一些广泛使用的设计技术，并展示很多用来实现 vector 的程序设计技术。我们的目标是向你展示一些主要的标准库功能是如何实现的，以及它们如何与硬件相配合。我们坚持认为一个工匠必须了解他的工具，而不是仅仅把工具当作“有魔力的东西”。

对于程序员来说，总是会对某些主题比对其他主题更感兴趣。但是，我们建议你不要预先判断你需要什么（你怎么知道你将来会需要什么呢？），至少每一章都要浏览一下。如果你学习本书是

作为一门课程的一部分，你的老师会指导你如何选择学习内容。

我们的教学方法可以描述为“深度优先”，也可以描述为“具体优先”和“基于概念”。首先，我们快速地（相对快速的，从第 1 ～ 11 章）将一些编写小的实用程序所需的技巧提供给你。在这期间，我们还简明扼要地提出很多工具和技术。我们着重介绍简单具体的代码实例，因为相对于抽象概念，人们能更快地领会具体实例，这是大多数人的学习方法。在最初阶段，你不应期望理解每个小的细节。特别是你会发现，对刚刚还工作得好好的程序只是稍加改动，便会呈现出“神秘”的效果。尽管如此，你还是要尝试一下！并且，请完成我们提供的简单练习和习题。请记住，在学习初期你只是没有掌握足够的概念和技巧来准确判断什么是简单的，什么是复杂的。请等待一些惊奇的事情发生，并从中学习吧。

我们会快速通过这个初始阶段——我们想尽可能快地带你进入编写有趣程序的阶段。有些人可能会质疑，“我们的进展应该慢些、谨慎些，我们应该先学会走，再学跑！”但是你见过婴儿学习走路吗？婴儿确实是在学会平稳地慢慢走路之前就开始自己学着跑了。同样，你可以先勇猛向前，偶尔摔一跤，从中获得程序设计的感觉，然后再慢下来，获得必要的精确控制能力和准确的理解。你必须在学会走之前就开始跑！

你不要投入大量精力试图学习一些语言细节或技术的所有相关内容。例如，你可以熟记所有 C++ 的内置类型及其使用规则。你当然可以这么做，而且这么做会使你觉得自己很博学。但是，这不会使你成为一名程序员。如果你学习中略过一些细节，将来可能偶尔会因为缺少相关知识而被“灼伤”，但这是获取编写好程序所需的完整知识结构的最快途径。注意，我们的这种方法本质上就是幼儿学习母语的方法，也是教授外语时最有效的方法。有时你不可避免地被难题困住，我们鼓励你向授课老师、朋友、同事、辅导员以及导师等寻求帮助。请放心，在前面这些章节中，所有内容本质上都不困难。但是，很多内容是你所不熟悉的，因此最初可能会感觉有点难。

随后，我们向你介绍一些入门技巧，来打牢你的知识和技能基础。我们通过实例和习题来强化你的理解，为你提供程序设计的概念基础。

我们重点强调思想和原理。思想能指导你求解实际问题——可以帮助你知道，在什么情况下，问题的求解方案是好的、合理的。你还应该理解这些思想背后的原理，从而理解为什么要接受这些思想，为什么遵循这些思想会对你和使用你的代码的用户有帮助。没有人会满意“因为事情就是这样的”这种解释。更重要的是，如果真正理解了思想和原理，你就能将已有的知识拓展到新的情况上，能用新的方法将思想和工具结合起来解决新的问题。知其所以然是学会程序设计技巧所必需的。相反，仅仅不求甚解地记住大量规则和语言特性有很大局限性，是错误之源，也是在浪费时间。我们认为你的时间很珍贵，尽力不浪费它。

我们把很多 C++ 语言层面的技术细节放在了附录中，你可以随时按需查找。我们假定你有能力查找到你需要的信息，你可以借助索引和目录来查找信息。不要忘了编译器和互联网也有在线帮助功能。但要记住，要对所有互联网资源保持足够高的怀疑，直至你有足够的理由相信它们。因为很多看起来很权威的网站实际上是由程序设计新手或者想要出售什么东西的人建立的。而另外一些网站，其内容都是过时的。我们在支持网站 www. stroustrup. com/Programming 上列出了一些有用的网站链接和信息。

请不要过于急切地期盼“实际的”例子。我们理想的实例都是能直接说明一种语言特性、一个概念或者一种技术的简短代码。很多现实世界中的实例比我们给出的实例要凌乱得多，而且所能展示的知识也不如我们的实例更多。数十万行规模的成功的商业程序中所采用的技术，我们用十几个 50 行规模的程序就能展示出来。最快地理解现实世界程序的途径是好好研究基本原理。

另一方面，我们不会用“聪明可爱的风格”来阐述我们的观点。我们假定你的目标是编写供他

人使用的实用程序，因此书中给出的实例要么是用来说明语言特性，要么是从实际应用中提取出来的。我们的叙述风格都是用专业人员对（将来的）专业人员的那种语气。

本书内容顺序的安排

讲授程序设计有很多方法。很明显，我们不赞同“我学习程序设计的方法就是最好的学习方法”这种流行的看法。为了方便学习，我们较早地提出一些仅仅几年前还是先进技术的内容。我们的设想是，本书内容的顺序完全由你学习程序设计过程中遇到的问题来决定，随着你对程序设计的理解和实际动手能力的提高，一个主题一个主题地平滑向前推进。本书的叙述顺序更像一部小说，而不是一部字典或者一种层次化的顺序。

一次性地学习所有程序设计原理、技术和语言功能是不可能的。因此，你需要选择其中一个子集作为起点。一般来说，一本教材或一门课程应该通过一系列的主题子集来引导学生。我们认为选择适当的主题并给出重点是我们的责任。我们不能简单地罗列出所有内容，必须做出取舍；在每个学习阶段，我们选择省略内容与选择保留内容是同样重要的。

作为对照，这里列出我们决定不采用的教学方法（仅仅是一个缩略列表），可能对你有用：

- C 优先：用这种方法学习 C++ 完全是浪费学生的时间，学生能用来求解问题的语言功能、技术和库比所需的要少得多，这样的程序设计实践很糟糕。与 C 相比，C ++ 能提供更强的类型检查，对新手来说有更好的标准库，以及用于错误处理的异常机制。
- 自底向上：学生本该学习好的、有效的程序设计技巧，但这种方法分散了学生的注意力。学生在求解问题过程中所能依靠的程序设计语言和库方面的支持明显不足，这样的程序设计实践质量很低、毫无用处。
- 如果你介绍某些内容，就必须介绍它的全部：这实际上意味着自底向上方法（一头扎进涉及的每个主题，越陷越深）。这种方法硬塞给初学者很多他们并不感兴趣，而且可能很长时间内都用不上的技术细节，令他们厌烦。这样做毫无必要，因为一旦学会了程序设计，你完全可以自己到手册中查找技术细节。这正是手册的用途，如果用来学习基本概念就太可怕了。
- 自顶向下：这种方法对一个主题从基本原理到细节逐步介绍，倾向于把读者的注意力从程序设计的实践层面上转移开，迫使读者一直专注于上层概念，而没有任何机会实际体会这些概念的重要性。例如，如果你没有实际体验编写程序是那么容易出错，而修正一个错误是那么困难，你就无法体会到正确的软件开发原理。
- 抽象优先：这种方法专注于一般原理，保护学生不受讨厌的现实问题限制条件的困扰，这会导致学生轻视实际问题、语言、工具和硬件限制。通常，这种方法基于“教学用语言”——一种将来不可能实际应用的语言，有意将学生与实际的硬件和系统问题隔绝开的语言。
- 软件工程理论优先：这种方法和抽象优先的方法具有与自顶向下方法一样的缺点：没有具体实例和实践体验，你无法体会到抽象理论的价值和正确的软件开发实践技巧。
- 面向对象先行：面向对象程序设计是组织代码和开发工作的最好方法，但并不是唯一有效的方法。特别是，以我们的体会，在类型系统和算法式程序设计方面打下良好的基础，是学习类和类层次设计的前提条件。本书确实在一开始就使用了用户自定义类型（有人称之为“对象”），但我们直到第 6 章才介绍如何设计一个类，而直到第 12 章才介绍类层次。
- “相信魔法”：这种方法只是向初学者展示强有力的工具和技术，但不介绍其下蕴含的技术和功能。这让学生只能去猜这些工具和技术为什么会有这样的表现，使用它们会付出多大代价，以及它们合理的应用范围，学生通常都会猜错！这会导致学生过分刻板地遵循相似

的工作模式，成为进一步学习的障碍。

当然，我们不会断言这些我们没有采用的方法毫无用处。实际上，在介绍一些特定的内容时，我们使用了其中一些方法，学生能体会到这些方法在一些特殊情况下的优点。但是，当学习程序设计是以实用为目标时，我们不把一些方法作为一般的教学方法，而是采用其他方法：主要是具体优先和深度优先方法，并对重点概念和技术加以强调。

程序设计和程序设计语言

我们首先介绍程序设计，把程序设计语言作为一种工具放在第二位。我们介绍的程序设计方法适用于任何通用的程序设计语言。我们的首要目的是帮助你学习一般概念、原理和技术，但是不能孤立地学习这些内容。例如，不同程序设计语言在语法细节、程序设计思想的表达及工具支持等方面各不相同。但对于编写无错代码的很多基本技术，如编写逻辑简单的代码（参见第 5 章和第 6 章），构造不变式（参见 9.4.3 节），以及接口和实现细节分离（参见 9.7 节和 14.1 节～ 14.2 节）等，不同程序设计语言则差别很小。

程序设计技术的学习必须借助于一门程序设计语言，设计、组织代码和调试等技巧是不可能从抽象理论中学到的。你必须用某种程序设计语言编写代码，从中获取实践经验。这意味着你必须学习一门程序设计语言的基础知识。这里说“基础知识”，是因为花几个星期就能掌握一门主流实用程序设计语言全部内容的日子已经一去不复返了。本书讲述的与 C++ 语言相关的内容只是它的一个子集，即与编写高质量代码关系最紧密的那部分内容。而且，我们所介绍的 C++ 特性都是你肯定会用到的，因为这些特性要么是出于逻辑完整性的要求，要么是 C++ 社区中最常见的。

可移植性

编写 C++ 程序运行于多种平台是很常见的。一些重要的 C++ 应用甚至运行于我们闻所未闻的平台上！我们认为可移植性和对多种平台架构与操作系统的利用是非常重要的。

本质上，本书的每个例子不仅是 ISO 标准 C++ 程序，而且是可移植的。除非特别指出，本书的代码都能运行于任何一种 C++ 实现，并且确实已经在多种计算机平台和操作系统上测试通过了。

不同系统编译、连接和运行 C++ 程序的细节各不相同，如果每当提及一个实现问题时，就介绍所有系统和所有编译器的细节，是非常单调乏味的。我们在附录 C 中给出了 Windows 平台 Visual Studio 和 Microsoft C ++ 入门的最基本的信息。

如果你在使用任何一种流行的，但相对复杂的集成开发环境（Integrated Development Environment，IDE）时遇到了困难，我们建议你尝试命令行工作方式，它极其简单。例如，下面给出的是用 GNU C++ 编译器，在 UNIX 或 Linux 系统中编译、连接和执行一个包含两个源文件 my_file1.cpp 和 myfile2.cpp 的简单程序所需的全部命令：

```
c++ -o my_program my_file1.cpp my_file2.cpp
./my_program
```

就这么简单，这真的就是全部。

程序设计和计算机科学

程序设计就是计算机科学的全部吗？答案当然是否定的！我们提出这一问题的唯一原因就是确实曾有人将其混淆。本书会简单涉及计算机科学的一些主题，如算法和数据结构，但我们的目标还

是讲授程序设计：设计和实现程序。这比广泛接受的计算机科学的概念更宽，但也更窄：

- 更宽，因为程序设计包含很多专业技巧，通常不能归类于任何一种科学。
- 更窄，因为对我们涉及的计算机科学的内容而言，我们没有系统地给出其基础。

本书的目标是作为计算机科学课程的一部分（如果成为一个计算机科学家是你的目标的话），作为软件构造和维护领域的第一门基础课程（如果你希望成为一个程序员或者软件工程师的话），是更大的完整系统的一部分。

本书自始至终都依赖计算机科学，我们也强调基本原理，但我们是以理论和经验为基础来讲授程序设计，是把它作为一种实践技能，而不是一门科学。

创造性和问题求解

本书的首要目标是帮助你学会用代码表达你的思想，而不是教你如何获得这些思想。沿着这样一个思路，我们给出很多实例，展示如何求解问题。每个实例通常先分析问题，随后对求解方案逐步求精。我们认为程序设计本身是问题求解的一种描述形式：只有完全理解了问题及其求解方案，你才能用程序来正确地表达它；而只有通过构造和测试一个程序，你才能确定你对问题和求解方案的理解是不是完整的、正确的。因此，程序设计本质上是理解问题和求解方案工作的一部分。但是，我们的目标是通过实例来说明这一切，而不是通过“布道”或是提供问题求解的详细“处方”。

反馈方法

我们认为不存在完美的教材，个人的需求总是差别很大的。但是，我们愿意尽力使本书和辅助材料更接近完美。为此，我们需要大家的反馈，脱离读者是不可能写出好教材的。请大家给我们发送反馈报告，包括内容错误、排版错误、含混的文字和缺失的解释等。我们也欢迎有关更好的习题、更好的实例、增加内容和删除内容等建议。大家提出的建设性的意见会帮助将来的读者，我们会将勘误表张贴在支持网站上：www. stroustrup. com/ Programming。

作者简介

你也许有理由问：“是一些什么人想要教我程序设计？”那么，下面是作者简介。我（即 Bjarne Stroustrup，如图 0-1 所示）和劳伦斯“皮特”彼得森（Lawrence“Pete”Petersen）合著了本书。我还设计并讲授了面向大学生初学者（一年级学生）的课程，这门课程是与本书同步开发的，以本书的初稿作为教材。

图 0-1　Bjarne Stroustrup

我是 C++ 语言的设计者和最初的实现者。在过去的大约 30 年间，我使用 C++ 和许多其他程序设计语言进行过各种各样的程序设计工作。我喜欢那些用在富有挑战性的应用（如机器人控制、绘图、游戏、文本分析以及网络应用）中的优美而又高效的代码。我教过能力和兴趣各异的人设计、程序设计和 C++ 语言。我是 ISO 标准组织 C++ 委员会的创建者，现在我是该委员会语言演化工作组的主席。

这是我第一本入门级的书。我编著的其他书籍，如 *The C++ Programming Language* 和 *The Design and Evolution of C++* 都是面向有经验的程序员的。

图 0-2 Lawrence“Pete”Petersen

我出生于丹麦奥尔胡斯的一个蓝领（工人阶级）家庭，在家乡的大学获得了数学与计算机科学硕士学位。我的计算机科学博士学位是在英国剑桥大学获得的。我为美国电话电报公司（AT&T）工作了大约 25 年，最初在著名的贝尔实验室的计算机科学研究中心——UNIX、C、C++ 及其他很多东西的发明地，后来在 AT&T 研究实验室。

我现在是美国国家工程院的院士，ACM 会士研究员和 IEEE 会士研究员，贝尔实验室院士和 AT&T 院士。我获得了 2005 年度 Sigma Xi 科学研究社区科学成就 William Procter 奖，我是首位获得此奖的计算机科学家。

2010 年，我获得丹麦 . 奥尔胡斯大学最老也是最有声望的奖项 Rigmor og Carl Holst-Knudsens Vrdens kapspris，这奖项是提供给与该校相关人士的科学成就。2013 年，我被俄罗斯圣彼得堡的信息技术、力学和光学（ITMO）国立研究大学授予计算机科学荣誉博士学位。

至于工作之外的生活，我已婚并有两个孩子，一个是医生，另一个目前是博士后。我喜欢阅读（包括历史、科幻、犯罪及时事等各类书籍），还喜欢各种音乐（包括古典音乐、摇滚、蓝调和乡村音乐）。和朋友一起享受美食是我生活中必不可少的一部分，我还喜欢访问世界各地有趣的地方和人。为了能够享受美食，我还坚持跑步。

关于我的更多信息，请浏览我的网站：www. research. att. com/ ～ bs 和 www. cs. tamu. edu/ people/ faculty/bs。特别地，你可以在那里找到我名字的正确发音。

在 2006 年末，Pete（如图 0-2 所示）如此介绍他自己：“我是一名教师。将近 20 年来，我一直在德州农工大学讲授程序设计。我已 5 次被学生选为优秀教师，并于 1996 年被工程学院的校友会选为杰出教师。我是 Wakonse 优秀教师计划的委员和教师发展研究院研究员。

作为一名陆军军官的儿子，我的童年是在不断迁移中度过的。在华盛顿大学获得哲学学位后，我作为野战炮兵官员和操作测试研究分析员在军队服役了 22 年。1971—1973 年期间，我在俄克拉荷马希尔堡讲授野战炮兵军官的高级课程。1979 年，我帮助创建了测试军官的训练课程，并在 1978—1981 年及 1985—1989 年期间在跨越美国的 9 个不同地方以首席教官的身份讲授这门课程。

1991 年我组建了一个小型的软件公司，生产供大学院系使用的管理软件，直至 1999 年。我的兴趣在于讲授、设计和实现供人使用的实用软件。我在佐治亚理工大学获得了工业管理学硕士学位，在德州农工大学获得了教育管理学硕士学位。我还从 NTS 获得了微型计算机硕士学位。我的信息和运营管理学博士学位是在德州农工大学获得的。

我的妻子芭芭拉和我都生于德州的布莱恩。我们喜欢旅行、园艺和招待朋友；我们花尽可能多的时间陪我们的儿子们和他们的家人，特别是我们的几个孙子孙女，安吉丽娜、卡洛斯、苔丝、埃弗里、尼古拉斯和乔丹。”

令人悲伤的是，Pete 于 2007 年死于肺癌。如果没有他，这门课程绝对不会取得成功。

附言

很多章都提供了一个简短的“附言”，试图给出本章所介绍内容的全景描述。这样做是因为我们意识到，知识可能是（而且通常就是）令人畏缩的，只有当完成了习题、学习了更多的章节（应用了本章中提出的思想）并进行了复习之后才能完全理解。不要恐慌，放轻松，这是很自然的，也是可以预料的。你不可能一天之内就成为专家，但可以通过学习本书逐步成为一名相当称职的程序员。在学习过程中，你会遇到很多知识、实例和技术，很多程序员已经从中发现了令人激动的和有趣的东西。