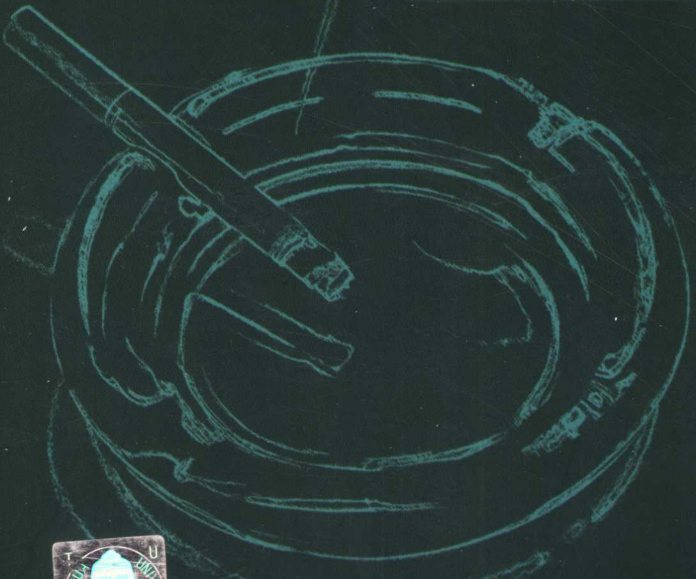
THOMSON

Object-Oriented Programming: Using C++ for Engineering and Technology

面向对象编程：工程和技术人员的C++语言

（影印版）



Goran Svenk 著

清华大学出版社

面向对象编程：
工程和技术人员的C++语言（影印版）

Object-Oriented Programming:
Using C++ for Engineering and Technology

Goran Svenk 著

清华大学出版社
北京

EISBN:0-7668-3894-3
Object-Oriented Programming:Using C++ for Engineering and Technology
Goran Svenk

图书在版编目（CIP）数据

面向对象编程：工程和技术人员的 C++语言=Object-Oriented Programming: Using C++ for Engineering and Technology/(加)斯万克著. —影印版. —北京：清华大学出版社，2003.10
ISBN 7-302-07216-7

I. 面…　II. 斯…　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 080087 号

出 版 者：	清华大学出版社 http://www.tup.com.cn **社总机**：010-62770175	**地　址**：	北京清华大学学研大厦
		邮　编：	100084
		客户服务：	010-62776969

责任编辑：冯志强
封面设计：付剑飞
印 刷 者：清华大学印刷厂
发 行 者：新华书店总店北京发行所 \ 清华大学出版社出版发行
开　本：185×260　**印张**：24.25　**字数**：603 千字
版　次：2003 年 10 月第 1 版　2003 年 10 月第 1 次印刷
书　号：ISBN 7-302-07216-7/TP・5251
印　数：1～3000
定　价：39.00 元

前　言

十余年来，C++已经成为了最流行、应用最广的编程语言之一。它被广泛地应用于工业和商业计算机应用程序开发的不同领域。C++和它的前身 C 已经成为工程技术的不同领域，如控制系统、通信、计算机辅助设计和嵌入式系统的主流编程语言。据估计，全球有几百万 C++程序员。1998 年通过的 C++标准使 C++的讲授、学习与程序开发更为简单。

本书适合的读者

本书适用于完成了使用 C 语言过程化程序设计课程的工程学、工程技术学、计算机科学和计算机研究技术专业的学生。书中包括许多电子工程的实例，因此对于电子工程或电子技术专业的学生更适合。本书适用于中级水平的程序员；同时，本书也是为程序员、工程师或者任何熟悉 C 而又想学习 C++的人准备的。对于已经有了 C++编程经验的人而言，从书中也会发现 C++程序设计有用的最新发展。

关于本书

C++是一种通用的程序设计语言，可支持底层程序设计、过程（结构化）程序设计、面向对象程序设计和普通程序设计。因此，C++书籍的作者们在决定使用什么方法时面临着一个困难。编写 C++书籍时面临的一个常见的两难选择是：只用纯粹的面向对象的方法，还是用混合过程化编程与 OOP 以平衡两者的方法。在大多数的工程类学校中，学生在学习 C++课程之前，已经成功地完成了 C 语言过程化程序设计课程。但是，大多数的 C++教材却使用混合的方法，这样对于那些已经完成了 C 语言课程的学生而言，书中就包含了大量多余的内容，从而导致篇幅不必要地增长，妨碍学生学习新的面向对象程序设计的概念。

本书的作者向工程学学生讲授 C++已长达八年之久。他长久以来一直在寻找一本使用纯面向对象的方法，并包含不同工程学学科程序示例的 C++图书。由于没有找到一本这样的书，于是作者决心自己写一本书来满足这些需求。本书使用纯面向对象的方法来讲授C++，不包含对于已经完成了 C 语言课程的读者而言多余的内容。本书包括许多电学和计算机工程学科的实例。本书是基于 ANSI/ISO C++标准的 C++程序设计语言教程。

本书的结构

本书包括 12 章。介绍性章节（第 1～3 章）讨论了 C 和 C++的不同，以及 C++对 C 的

过程化程序设计的增强。其余各章（第 4～12 章）用丰富的程序示例展示了面向对象的概念。各章按如下方式组织：

- **第 1 章，从 C 过渡到 C++**。阐述 C 和 C++之间的基本区别，并讨论名称空间和 C++的输入/输出。
- **第 2 章，C++函数的增强功能**。就 C++函数相对于 C 函数的优势进行解释，讨论函数功能的增强。
- **第 3 章，指针、引用与动态内存分配**。重点介绍指针和引用的使用，并且讨论 C++中动态内存分配技术及其实现方式。
- **第 4 章，类和对象**。解释面向对象的关键概念和技术，介绍 C++的扩展结构、类以及创建和销毁对象的机制。
- **第 5 章，类的高级议题**。解释向函数传递和从函数返回对象的过程，讨论副本构造函数、友元函数和友元类以及静态类成员、this 指针和常量成员函数。
- **第 6 章，运算符重载**。讨论运算符重载的所有内容，并且演示实现运算符重载的程序实例。
- **第 7 章，继承**。阐述实现继承的一些最重要的内容，如构造和销毁派生的类对象，使用多重直接和多重间接继承，重载和支配继承的类成员。
- **第 8 章，合成**。讨论并演示一些合成的实例，以及组合合成与继承的程序示例。
- **第 9 章，多态性和虚函数**。解释静态绑定与动态绑定之间的不同，阐述在实现运行时多态性时虚函数和抽象基类的重要性。
- **第 10 章，模板**。解释类模板、容器和迭代器，介绍 STL 库。
- **第 11 章，异常处理**。描述和演示 C++中的异常处理机制和工具。
- **第 12 章，文件 I/O**。讨论 C++中的文件 I/O 处理的步骤，并演示在处理顺序文件和随机存取文件时常用的一些 C++技术。

本书特色

本文使用纯粹的面向对象的方法来讲授 C++程序设计的精髓，同时也讨论了 C++对 C 的过程化程序设计的增强。本书可以用做学习工具书，也是一本有价值的参考书。它包括许多电学和计算机工程学专业的实例。另外还对使用 C++的软件工程和程序设计的许多方面进行了介绍。

书中每一章都分为若干节。为使本书正文的可读性更强，代码清单、图、表和重要的程序设计技巧都使用了不同于正文的版式。程序示例添加了行号以方便读者在文中找到特定的代码行。较短的代码段则直接插入到文中而未附加行号。

本书包含许多适合教学的特色：

- 在每章的起始处都列出了一些学习目标。
- 强调了程序设计技巧和重点提示。
- 每章中的程序示例展示了特定的程序设计概念和 C++工具。
- 案例研究解决各种工程学专业的典型问题。按照逻辑步骤使用伪代码和流程图来分

析问题并开发程序。

- 对每章的关键知识点都有小结。
- 每章末尾都有练习题以检验对该章所述内容的理解。其中一些问题需要分析、跟踪程序段，而另外一些则需要编写、修改代码或对代码段进行错误检查。本书最后提供了单数练习题的答案。
- 每章末尾的编程项目需要使用该章讨论的程序设计工具和技术来完整地解决典型的工程学问题。
- 相关网站提供了本书所有的程序示例以及案例研究的源代码文件。

补充材料

作为本书的补充，教师手册包含了程序设计项目和练习的解决方案。该手册还包含关于不同 C++编译器的必需的信息。

关于作者

作者 Goran Svenk 是 Seneca 学院技术系的教授，也是电子和计算机工程技术学院的软件工程专业的学术带头人。他在学院或大学讲授计算机科学和控制系统的不同课程超过了 18 年。同时负责一些程序设计语言(Pascal、C、C++、Visual Basic 和 Java)的课程设计与开发研究。

本书源代码可以从以下网站下载：

www.tupwq.net

PREFACE

For over a decade, C++ has been one of the most popular and widely used programming languages. It is used for computer applications development in a broad variety of fields in both industry and business. C++ and its predecessor, C, have been dominant programming languages in diverse areas of engineering and engineering technology; such as, control systems, telecommunications, computer-aided design, and embedded systems. Estimates suggest that there are several million C++ programmers worldwide. The C++ standard, ratified in 1998, has made it easier to teach, learn, and develop applications using C++.

INTENDED AUDIENCE

This book is ideal for students in engineering, engineering technology, computer science, or computer studies technology programs who have completed a procedural programming course using C language. It contains many practical examples from electrical engineering, and therefore relates well to students in electrical engineering or electronics technology programs. *Object-Oriented Programming: Using C++ for Engineering and Technology* is aimed specifically at intermediate-level programmers. This book is also intended for programmers, engineers, or anyone who is familiar with C and wants to learn C++. Those who already have had experience with C++ will find the latest developments in C++ programming useful.

ABOUT THIS BOOK

C++ is a general-purpose programming language that supports low-level programming, procedural (structured) programming, object-oriented programming (OOP), and generic programming. The authors of C++ books, therefore, have faced a difficult challenge when making a decision about the approach to use. A common dilemma when writing a C++ book is whether the book should present a pure object-oriented approach, or a hybrid approach balancing procedural programming with OOP. In most of the engineering schools, students are required to successfully complete procedural programming using C prior to taking a C++ course. Most of the C++ textbooks, however, use the hybrid approach and contain a significant amount of redundant material for those who have completed a C course. This results in an unnecessarily lengthy text that discourages students from learning new OOP concepts.

This textbook's author has taught C++ to engineering students for more than eight years. He has been searching for a C++ book that uses a pure object-oriented approach and contains program examples from a variety of engineering disciplines. Not being able to find such a textbook, the author decided to write a book that met these requirements. *Object-Oriented Programming:*

Using C++ for Engineering and Technology uses a pure object-oriented approach to teach C++ and avoids redundant material for those who have already completed a C course. The book also discusses the differences between C and C++. It contains many practical examples from the electrical and computer engineering disciplines. The text is based on the C++ programming language as defined by the ANSI/ISO C++ standard.

BOOK ORGANIZATION

The book is organized into 12 chapters. The introductory chapters (Chapters 1 through 3) discuss both the differences between C and C++ and the C++ enhancements to procedural programming in C. The remaining chapters (Chapters 4 through 12) present OOP concepts with a rich collection of program examples. The chapters are organized in the following manner:

- **Chapter 1, Moving from C to C++,** addresses the fundamental differences between C and C++ and discusses both namespaces and C++ input/output.
- **Chapter 2, C++ Function Enhancements,** explains the advantages of C++ functions over C functions and discusses C++ function enhancements.
- **Chapter 3, Pointers, References, and Dynamic Memory Allocation,** emphasizes the use of pointers and references and discusses the concept of dynamic memory allocation and its implementation in C++.
- **Chapter 4, Classes and Objects,** explains the key OOP concepts and terminology and introduces C++ expanded structures, classes, and mechanisms of creating and destroying class objects.
- **Chapter 5, Classes: Advanced Topics,** explains the procedures of passing and returning objects from functions and discusses copy constructors, ***friend*** functions and ***friend*** classes, ***static*** class members, the ***this*** pointer, and *const* member functions.
- **Chapter 6, Operator Overloading,** discusses all aspects of operator overloading and demonstrates practical examples of programs that implement operator overloading.
- **Chapter 7, Inheritance,** addresses the most important aspects of the implementation of inheritance; such as, constructing and destroying derived class objects, using multiple-direct and multiple-indirect inheritance, and overriding and dominating inherited class members.
- **Chapter 8, Composition,** discusses and demonstrates some practical examples of composition, as well as program examples that combine composition and inheritance.
- **Chapter 9, Polymorphism and Virtual Functions,** explains the difference between static and dynamic binding and addresses the importance of both ***virtual*** functions and abstract base classes when implementing run-time polymorphism.
- **Chapter 10,Templates,** explains class templates, containers and iterators, and introduces the STL library.
- **Chapter 11, Exception Handling,** describes and demonstrates the exception-handling

mechanisms and tools in C++.

- **Chapter 12, File I/O,** discusses the steps in C++ file I/O processing and demonstrates some commonly used C++ techniques when processing sequential and random-access files.

FEATURES

The text uses a pure object-oriented approach to teach the essentials of programming using C++. It also discusses the C++ enhancements to procedural programming in C. This text could be used as a learning tool, or as a valuable reference sourcebook. It includes many practical examples from the electrical and computer engineering disciplines. The text presents many aspects of software engineering and program design using C++.

Every chapter is divided into sections. To make the text more readable, code listings, figures, tables, and important programming tips are set off from the text using different graphic styles. Program examples are numbered for easy reference to specific lines in the text. Short code segments are inserted directly into the text without line numbers.

The book uses a wide spectrum of pedagogical features, such as

- **A set of chapter objectives and a table of contents** at the beginning of every chapter.
- **Programming tips and important notes** that are emphasized.
- **Program examples** in every chapter to exhibit specific programming concepts and C++ tools.
- **Case studies** that solve typical problems in a variety of engineering disciplines. The problems are analyzed and the programs are developed in logical steps using pseudocode and flowcharts.
- **Chapter summaries** of key points.
- **End-of-chapter review questions and problems** follow every chapter to check the understanding of the material covered. Some problems require an analysis and tracing of program segments, while the others require writing, modifying, or troubleshooting code segments. Answers to the odd-numbered questions are provided at the end of the text.
- **Programming projects** at the end of every chapter that require complete solutions to typical engineering problems using the programming tools and techniques discussed in that particular chapter.
- An accompanying ***CD-ROM*** contains the C++ compiler (Microsoft Visual C++, Learning Edition) and source code files for all program examples and case studies in the text.

The following conventions are used to make the text more readable

- Important terms and definitions are shown using a bolded text font (e.g.**multiple inheritance, abstract class**).
- C++ keywords and standard library functions are shown in a bolded and italicized font

(e.g., ***new, static, push_back()***).

- User-defined identifiers and functions are shown in Courier font (e.g., ***amplifier, get-Current()).***
- Keywords with the prefix "non"- will be shown in italicized text font. Italicized text font is also used to emphasize important concepts or words (e.g., ***non-friend, parent class***).

SUPPLEMENTS

To supplement this text, an Instructor's Manual, (ISBN#:0-7668-3895-1) contains solutions to the programming projects and exercises. This manual also includes necessary information about different C++ compilers.

ABOUT THE AUTHOR

The author is a professor in the Faculty of Technology at Seneca College and a group facilitator for software engineering in the School of Electronics and Computer Engineering Technology. He has taught a variety of courses in computer science and control systems at colleges and universities for over 18 years. He has also been responsible for course and curriculum development for a number of programming languages (Pascal, C, C++, Visual Basic, and JAVA).

ACKNOWLEDGMENTS

I would like to thank Greg Clayton and Michelle Ruelos Cannistraci , my editors, and all the crew at Delmar Learning—Larry Main, Christopher Chien, David Arsenault, and Jennifer Luck—for their support and constant encouragement. It has been a pleasure to work with them.

I would also like to express my gratitude to Professor Len Klochek and Professor Martyn McKinney, my colleagues from Seneca College, for their extraordinary effort and contributions in making this a better book. Professor Klochek's technical review of the manuscript and his suggestions were exceptionally helpful to me when preparing the final version of this book.

I would also like to thank the following reviewers for their very constructive comments:

Richard L. Henderson, DeVry University, Kansas City,MO
Deneil Lutter, DeVry University, Kansas City,MO
Bud Berges, DeVry University, Calgary,Alberta
Len Klochek, Seneca College,Toronto, Ontario

DEDICATION

To My Parents, Djuja and Anton Svenk

CONTENTS

Chapter 1

MOVING FROM C TO C++

OBJECTIVES

- To understand the differences between C and C++
- To be able to use C++ input/output and formatting
- To understand the ***namespace*** mechanism and to be able to define and use namespaces
- To become familiar with the C++ standard

CHAPTER CONTENTS

INTRODUCTION

Although C++ evolved from C, it is considered to be a different programming language. Many programmers also consider C++ as a superset of C. C++ adds new, enhanced features to some important C concepts such as functions, pointers, and structures.Unlike C, which is a pure structured (procedural) language, C++ also provides another approach to programming called **object-oriented programming**. Objectoriented programming is more efficient when dealing with large, complex projects and requires a different way of visualizing solutions to programming problems.

Instead of C's standard I/O functions, C++ uses different tools to get input and produce output. C++ also provides a new tool called a ***namespace*** to prevent errors that may occur when linking multiple files containing source code into one application. This chapter will first address the fundamental differences between C and C++. It will then discuss the concepts of C++ nput/output and namespaces.

1.1 DIFFERENCES BETWEEN C AND C++

C has been used for almost three decades and has been the most popular structured programming language in engineering. Its structured approach, however, limits its use in large and complex projects. The main reasons for expanding C and creating C++ were the following:

- To provide a new approach to programming (object-oriented programming) that would overcome the limits of C's structured approach when dealing with large and complex programming problems.
- To design new programming tools that would help programmers write code quickly and efficiently, as well as help improve maintainability of large projects.
- To design a more rigid programming language that would require programmers to follow certain strict rules, thus reducing chances for making errors.
- To create a highly extendable programming language.
- To enhance some important C concepts such as functions, pointers, and structures and make these tools even more powerful.

These five reasons for creating C++ are fundamental to the differences between C and C++. When problems in programming exceed a certain size, C's structured approach makes such programs hard to design and maintain. C++'s object-oriented approach is much more efficient when dealing with large and complex programming problems. Using object-oriented tools when developing a large project decreases the program's development time and improves its maintainability.

C is less rigid than C++. For example, C does not require programmers to specify a function type or function arguments when writing function prototypes. There are many other examples of C's lack of rigidity that will be discussed in this book. A flexible programming language like C allows the compiler to make certain decisions for programmers. This is a disadvantage because the compiler may make an incorrect guess as to the programmer's intention, leading to increased errors. C++ is much more rigid than C and forces programmers to follow very strict rules when writing code.

C++ is much larger (provides more programming tools) and more complex than C. C may also be defined as a subset of C++. Any C++ compiler can therefore compile C programs as well, with very few coding changes. The relationship between C and C++ can be graphically represented, as shown in Figure 1.1. A small circle representing C is located almost entirely within a large circle representing C++. Only a small component of C does not also belong to C++.

C is very efficient when solving small and simple programming problems. C++ is the language of choice when dealing with large and complex problems. There is no reason to use C++'s advanced tools in small and simple programs—it may even needlessly increase their complexity.