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

Goran Svenk 著
马海军 段晓勇 译

清华大学出版社

面向对象编程:

工程和技术人员的C++语言

Goran Svenk 著

马海军 段晓勇 译

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

北京市版权局著作权合同登记号　图字 01-2003-2559 号

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/（加）思万克著；马海军，段晓勇译.—北京：清华大学出版社，2003.8
书名原文：Object-Oriented Programming：Using C++ for Engineering and Technology
ISBN 7-302-06618-3

Ⅰ.面… Ⅱ.①思… ②马… ③段… Ⅲ.①面向对象语言—程序设计 ②C 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 056798 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社总机：010-62770175　　**客户服务**：010-62776969
责任编辑：冯志强
封面设计：付剑飞
印 刷 者：北京鑫丰华彩印有限公司
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**：22.25　**字数**：551 千字
版　　次：2003 年 8 月第 1 版　2003 年 8 月第 1 次印刷
书　　号：ISBN 7-302-06618-3/TP・5103
印　　数：1～4000
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++的教授、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的增长，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教授 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教授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教授 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。本书可以用作学习工具书，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。他在学院或大学教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(Rascal、C、C++、Visual Basic 和 Java)的课程设计与开发研究。

Goran Svenk
多伦多，2002

作 者 简 介

Goran Svenk 在电子和计算机工程领域具有 21 年的产业与教育经验，并且编著了大量图书和文章。他从 1983 年起就在各类学院执教，现在是加拿大安大略省 Toronto 的 Seneca College 的全职教授，从 1997 年起他就在此工作，并且开发了许多计算机编程领域的课程。

本书源代码可以从以下网站下载：

www.wqtup.com.cn

感谢本书的编辑 Greg Clayton 和 Michelle Ruelos Cannistraci，以及 Delmar Learning 的全体成员——Larry Main、Christopher Chien、David Arsenault 和 Jennifer Luck——对我的支持和不断鼓励。与他们一同工作使我倍感欢乐。

我还要对 Len Klochek 教授、Martyn McKinney 教授和 Seneca 学院的同事表达我的感激之情，以感谢他们对本书所做出的非凡的努力和贡献。在准备本书的最终版本时，Klochek 教授对手稿进行的技术审校和提出的建议对我有很大的帮助。

我还想为以下审校人员提出的建设性的意见表示感谢：

Deneil Lutter, DeVry University, Kansas City,MO

Bud Berges, DeVry University, Calgary,Alberta

Len Klochek, Seneca College,Toronto, Ontario

献　　辞

献给我的父母 Djuja 和 Anton Svenk。

目　录

第 1 章　从 C 过渡到 C++

学习目标

- 理解 C 和 C++的不同之处
- 能够使用 C++的输入/输出和语法格式
- 理解名称空间的机制，并能够定义和使用名称空间
- 熟悉 C++标准

内容简介

虽然 C++是从 C 发展来的，但是它仍然被认为是一个不同的程序设计语言。许多编程人员也把 C++当作 C 的一个超集。C++对 C 的一些重要概念，如函数、指针和结构添加了新的增强功能。与 C 这一纯粹的结构化（过程化）语言不同，C++还提供了另外一种称为面向对象的编程方法来进行程序设计。在处理大型的复杂对象时，面向对象的程序设计更加有效，而且它要求用一种不同的直观的方法去解决程序设计中的问题。

C++使用一些不同的工具取代 C 的标准输入/输出函数来获取输入、生成输出。C++还提供了一个名为名称空间的新工具，用以防止把包括源代码的多个文件链接到一个应用程序时可能发生的错误。本章将首先介绍 C 和 C++的主要不同，然后再讨论 C++输入/输出和名称空间的概念。

1.1　C 和 C++的不同之处

C 被使用了近 30 年，它已经成为了工程技术领域最流行的结构化语言。然而，它的结构化方法限制了它在大型的复杂工程中的使用。扩展 C 以及创建 C++的主要原因如下：

- 为了提供一个新的程序设计方法（即面向对象的程序设计方法），从而在处理大型的复杂程序设计问题时克服 C 的结构化方法的局限。
- 为了设计新的编程工具帮助编程人员快速有效地编写代码，同时也有利于改善大型工程的可维护性。
- 为了设计一个更严格的程序设计语言，要求编程人员遵守一定的严格规则，以减少犯错的机会。
- 为了创建一个高度可扩展的程序设计语言。
- 为了增强一些重要的 C 工具，如函数、指针和结构，以使这些工具更强大。

创建 C++的 5 个原因就是 C 和 C++的主要不同之处。

当程序设计中的问题超过一定规模时，C 的结构化方法会使这样的程序难以设计和维

护。C++的面向对象方法在处理大型复杂的程序设计问题时则更为有效。开发一个大型项目时，使用面向对象的工具减少了程序的开发时间，并且可以提高程序的可维护性。

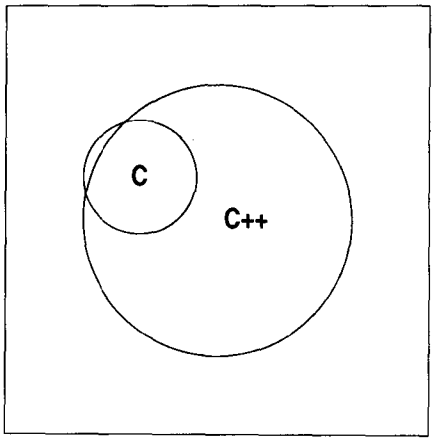

图 1.1 C 和 C++关系的图示

C 不如 C++严格。例如，C 在编写函数原型时不需要程序指定函数类型或函数参数。此外，书中还讨论了很多其他 C 缺乏严格性的例子。像 C 这样灵活的程序设计语言允许编译器为编程人员做出某些决定。这是一个缺陷，因为编译器做出的猜测可能与编程人员的意向相左，从而导致错误的增加。C++则比 C 严格得多，它迫使编程人员在编写代码时遵守非常严格的规则。

C++比 C 更大（即提供更多的编程工具）、更复杂。C 也可以被定义为 C++的一个子集。因此，编写的代码几乎不用变化，C++编译器就可以编译 C 程序。C 和 C++的关系可以用图 1.1 来表示。代表 C 的小圆几乎全部位于代表 C++的大圆之中。C 只有一小部分不属于 C++。

在解决一些小型的简单程序设计问题时，C 是非常有效的。而 C++是解决大型的复杂问题时所选择的语言。我们不应该在小型的简单程序中使用 C++的高级工具——这样甚至会增加它们不必要的复杂性。

1.1.1 C++标准和编译器

今天，许多不同的 C++编译器在业界和学校中使用，常用的有不同版本的 Borland C++、Microsoft Visual C++和 Borland C++ Builder。其中一些编译器可能比目前的发布于 1988 年的 ANSI/ISO C++标准更陈旧。因此，编程人员在试图使用一些他们所用的编译器不支持的标准库和高级 C++工具时会遇到问题。所以，在设计程序之前先检查这些编译器是否使用 C++标准进行编译是很重要的。

C++标准文档的电子复制件（标题为 Programming Languages——C++，索引号为 ISO /IEC 14882:1998）可以从下列网址订阅：

http://www.ansi.org/

需要注意的是，本书代码是使用当前的 ANSI/ISO C++标准编译的。因此，如果你使用旧的或者非标准的编译器，一些程序示例可能无法编译。但是在大多数情况下，只需稍微修改就可以成功编译这些程序。必要的编译器信息将在合适的地方提供给读者。

1.1.2 C++注释

在代码中写上注释是一个良好的编程习惯。注释不但可以提高程序的可读性，而且也有助于程序的维护。C++提供了一个新的注释方式，每个注释行以两个正斜线（//）开始并延续到该行的结尾。与 C 中的注释方式不同，C++的注释方式不使用任何特殊的标记来结束一个注释行或注释块。编译器会忽略以两个正斜线开始直至当前行行尾的文本。

C++也支持 C 的注释方式，它以/*来标记一个注释的开始，并用*/来标记该注释的结束。下列的代码段举例说明了这两种注释方式：

```
/* This is the old C commenting style, in which
multiple comment lines (a comment block) begin
with a forward slash followed by an asterisk and
end with an asterisk followed by a forward slash. */

//This is the new C++ commenting style, in which each
//line begins with two forward slashes without end symbols.
```

C++编程人员可以使用两种注释方式。C 的注释方式适用于较长的多行注释，而新的C++注释方式则适用于单行注释。

1.1.3　头文件

C++库是名为头文件（header file）的文件的集合，包含程序实体，如函数、数据类型和常量的定义和声明。#include 预处理程序指令后跟随一个头文件名放在程序的开始处，就可以把头文件的内容引入到该程序当中。预处理程序使用指令中指定的文件的副本来取代#include 指令。标准库头文件名要包含在尖括号中（<>）。

C 语言需要所有的头文件都有一个.h 扩展名，而 C++标准则不需要标准头文件有任何扩展名。下面的代码段演示了#include 指令在 C 和 C++中的不同。

```
#include <stdio.h>      //Includes C' s standard header file
#include <iostream>     //Includes C++' s standard header file
```

除使用标准库头文件之外，编程人员也可以创建自己的头文件，名为用户定义（user-defined）头文件。用户定义的头文件也应该有一个.h 的扩展名。#include 预处理程序指令也可以用来引入用户定义的头文件。与标准库头文件名包含在尖括号中不同，用户定义的头文件名应该包含在双引号内。下面的代码段举例说明了这种语法。

```
#include <iostream>     //Includes a standard header file
#include " myheader.h"  //Includes a user-defined header file
```

包含在双引号中的用户定义的头文件的文件名会使预处理程序首先在存储被编译的源文件的目录中搜索头文件。如果在这个目录中没有找到用户定义的头文件，预处理程序就继续在标准库头文件的指定路径中搜索。C++使用的搜索机制与运行程序的计算机系统的搜索机制是不同的。

注意： 附录 C 包含一个 C++标准库头文件及每个文件简要描述的列表。

1.1.4　数据类型

除了 C 的 5 种基本数据类型以外（char、int、float、double 和 void），C++还提供了一

个新的数据类型，名为 bool。bool 类型的值——也称为布尔值（Boolean values）——可以是 true 或 false。这些值常用在逻辑操作中。bool 类型的值可以被转换成 int 类型的值。关键字 true 和 false 被预定义为数值——true 为数值 1，false 为数值 0。下面的代码段演示了一个返回 bool 类型值的函数：

```
bool fun (float x, float y )  //Function returns true or false
{
     bool answer;             //Variable answer of bool type is instantiated
     if (x<y && x>0)
        answer=true;          //true is assigned to answer
     else
        answer=false;         //false is assigned to answer
     return answer;
}
```

这个函数实例化一个 bool 类型的 answer 变量。根据 if 条件的值，answer 变量被赋值为 true 或 false。函数返回 answer 的值。

创建表示 true 或 false 状态的变量时，用 C 中的 int 类型不如 bool 类型。使用 bool 值可增强程序的可读性，使其逻辑更清晰。除了提高程序的清晰度以外，bool 类型消耗的内存也比 int 类型少。

C 和 C++都允许在表达式或赋值语句中将一种类型的值与另一种类型的值混合使用。这种情况发生时，会使用自动数据类型转换（automatic data type conversion）的以下规则：

（1）不同类型的值首先被转换成表达式中使用的最大的数据类型，然后再计算表达式。

（2）在赋值语句右边的值被转换成左面的数据类型。

下面的代码段演示了自动数据类型转换的规则：

```
int a=2;
float b=3.12;
double c=1.9999, d;
d=a/b*c;   //Values of a and b are temporarily converted to double
a=c;       //Value of c is temporarily converted to int
```

a 和 b 的值首先被临时转换成 double（即表达式 a/b*c 中最大的类型），然后计算表达式。在 c 赋值给 a 之前，c 的值被临时转换成 int（赋值语句左边的数据类型）。转换的结果是变量 a 的值被置成 1，因为 1.9999 的小数部分被截去了。

在计算表达式和执行赋值语句之前，编译器会自动地执行数据类型转换。需要注意的是，变量的值只是为表达式的求值而被临时转换。存储在内存中的值仍然会保持不变。这些自动类型转换可能会成为逻辑错误的一个来源。C++编译器在每一次类型转换发生时都会警告编程人员。这些警告表明发生了一些不寻常的事件，从而帮助编程人员避免一些类型的逻辑错误（例如，通过使用类型强制转换）。需要注意的是，类型转换是编程人员请求的一个显式数据类型变化。下面的代码段演示了 C 方式的类型转换也可以用在 C++中：

```
int a=9, b=10;
float percent;
percent=a/b*100;        //assigns 0.0 to percent
```

```
//Use type cast to change a to float before performing division.
percent= (float) a/b*100; //assigns 90.0 to percent
```

在第一个赋值语句(a/b*100)右边的表达式使用了 3 个整数值，因而被作为 int 求值(整型除法 9/10=0)。表达式的值在赋值给 percent 之前被转换成 float（0.0）。第二个赋值语句中的表达式使用了一个显式强制类型转换，把 a 临时转换成 float。随后，变量 b 在实数除法（9.0/10.0 = 0.9）中通过编译器自动临时转换成 float。使用强制类型转换的结果是将一个正确的值赋给了 percent（90.0）。

除了 C 方式的强制类型转换外，C++还提供了下列的强制转换运算符：

- static_cast 用于标准类型转换
  例如：static_cast<float>（a）将 a 的类型转换为 float
- const_cast 用于强制转换常量
- reinterpret_cast 用于非标准类型的转换

C++类型转换运算符将在后面几章的程序示例中进行演示。

C 中的变量不必存储它声明的数据类型。与 C 不同，C++在数据类型检查方面更为严格。虽然 C++依然允许不同数据类型之间的混合或赋值，但是在一些情况下，类型转换必须在赋值之前使用。

1.1.5　解析作用域和变量声明

C 和 C++的变量可以声明为全局的（在函数定义之外，通常在 main( )之前）和局部的（在函数中）。关于局部声明，C 标准和 C++标准的不同之处如下所示：

- 标准 C 中的局部变量必须在表明函数定义的开始的花括号（{}）之后声明，或者在表明函数定义中一个块的开始的花括号之后声明。
- C++中的局部变量可以在函数体或块内的任何地方声明，只要它们在使用之前被声明即可。

C 和 C++都允许局部变量和全局变量同名。这两种语言也都允许两个或更多的同名局部变量在同一个函数内声明，只要它们不是在同一个块中声明就可以。

当在 C 程序中使用两个或更多同名变量时，只有最内层声明的局部变量可以在任何时间被访问。而当全局变量处于使用于全局变量同名的局部变量的函数或者块中的时候，该全局变量不能被访问。

C++提供了作用域解析运算符（scope resolution operator），表示为“::”，它用于访问与局部变量同名的全局变量。下面的代码段演示了作用域解析运算符的用法：

```
int x=1;                    //global variable
int main( )
{
    int x=2;                //local variable
    ::x=8;                  //assigns 8 to the global variable
    {                       //beginning of a block
        int x=3;            //new local variable whose scope
                            //is the inner block
          cout<<x;          //prints value of local variable (3)
```