HZ BOOKS

PEARSON
Addison Wesley

C++编程风格

C++ Programming Style

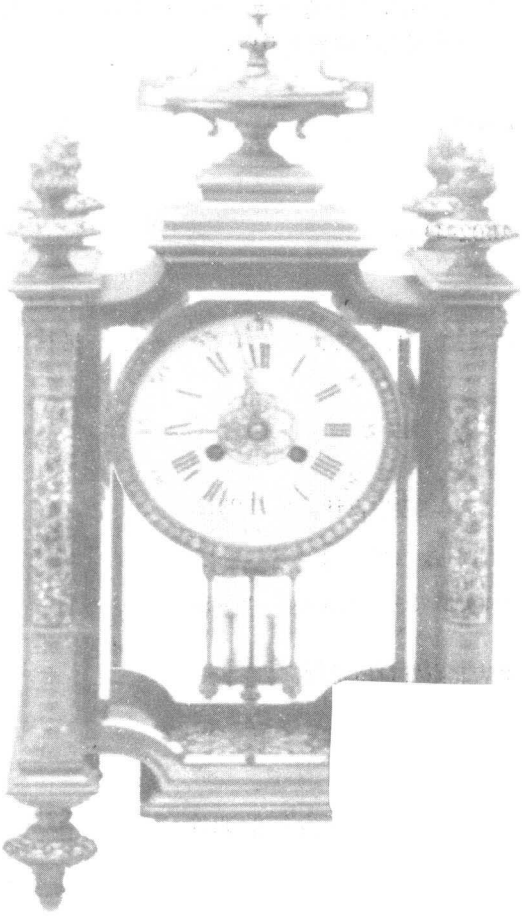
(美) Tom Cargill 著
聂雪军 译

机械工业出版社
China Machine Press

C++编程风格

C++ Programming Style

(美) Tom Cargill 著
聂雪军 译



机械工业出版社
China Machine Press

本书描述C++语言中较深层次的程序设计思想和使用方法，包含大量软件工程概念和设计模式，重点介绍大规模编程相关的内容，例如增加代码的可读性、可维护性、可扩展性以及执行效率等的方法。本书的示例代码都是从实际程序中抽取出来的，融入了作者的实际开发经验。讲解如何正确地编写代码以及避开一些常见的误区和陷阱，并给出了许多实用的编程规则，可快速提升读者的C++编程功力。

本书描述平实，示例丰富，适合有一定编程经验的计算机程序设计与开发人员参考。

本书版权登记号：图字：01-2002-4186

图书在版编目（CIP）数据

C++编程风格/（美）卡吉尔（Cargill，T.）著；聂雪军译．—北京：机械工业出版社，2007.1

书名原文：C++ Programming Style

ISBN 7-111-20363-1

Ⅰ.C…　Ⅱ.①卡…　②聂…　Ⅲ.C语言-程序设计　Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2006）第 136562 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：李东震

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2007 年 1 月第 1 版第 1 次印刷

186mm×240mm · 12 印张

定价：25.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

这是一本能够提升你的C++编程功力的书籍。

C++是一种功能强大的编程语言，这已是不争的事实。目前，许多关于C++的书籍都只是对语言本身进行了详细的介绍，而对于如何正确地使用C++语言却介绍得不多。这就好比我们拥有了一种强大的武器，却无法有效地发挥它的最大威力。例如，虚函数和动态绑定是C++的强大功能之一，但我们必须首先识别出正确的抽象，才能有效地使用虚函数；否则，盲目地使用虚函数反而会降低程序的性能。与其他的C++语言书籍相比，本书更像是一本通过C++来阐述软件工程思想和设计模式的书籍。书中所讲述的大多数问题都是与大规模编程相关的，重点是放在程序中不同组件之间的交互行为上，而不是诸如命名规则、注释风格等细节问题上。对程序员来说，实现某个功能并不是件难事，困难的是如何使设计和编码更为合理，这其中包括：代码的可读性、可维护性、可扩展性以及执行效率等。通过对本书的学习，你将会掌握正确的设计思想和使用C++的方法。

本书的最大特点就是理论与实际紧密相连。书中的示例代码都是从实际程序中抽取出来的。译者在翻译本书的时候，总有一种似曾相识的感觉：书中所提到的许多错误，在译者还是新手的时候都曾遇到过，因此能够与作者的分析和讲解产生共鸣。在任何一名新手的成长过程中，总是要经过痛苦的实践和许多不眠之夜，才能够积累一些经验和教训。而在本书中，作者将这些最宝贵的东西一并呈现给了读者。

本书的另一特点就是它的平实性。书中既没有华丽的辞藻，也没有尖锐的观点，而只是通过普通的叙述方法，将我们在编码过程中所遇到的问题加以剖析。细细读来，就好像是经验丰富的大师正在给新手们讲解如何正确地编写代码以及避开一些常见的误区和陷阱。

书中所给出的编程规则对于程序员设计和编码是很有帮助的。但是切记，任何规则都不是绝对的，在每一条规则后面都有着特定的使用条件。即使是同一条规则，在不同的环境中，所起的作用可能是截然不同的。我们应该灵活地使用这些规则，而不是生搬硬套。否则，规则将会变成教条，这就违背了作者的初衷。

译者深知“一份耕耘，一份收获”，因此在翻译过程中不敢有一丝懈怠。对每个不易理解的地方，译者总是仔细斟酌，反复推敲，尽最大努力去将那些精妙的思想用浅显的语言表达出来。然而，由于时间和水平有限，翻译中的疏漏之处在所难免，还望读者和同行不吝指正。

致谢

本书的完成首先要感谢好友王昕，是他促成了我与机械工业出版社的这次合作。参加本书翻译的还有李杨、吴汉平、徐光景、童胜汉、陈军、胡凯、李斌、张玮、陈红、刘红等。感谢华章分社陈冀康编辑对于我的信任和耐心，使我能够顺利地完成本书。感谢出版社其他编辑对本书的审阅和校对，感谢你们的细心。感谢妻子云兰和我们即将出世的孩子，感谢你们，你们是我永远的动力。感谢我的父母，你们一直都在默默地支持着我。

聂雪军

2006 年 5 月于北京

前　言

Kernighan 和 Plauger 的经典之作《*The Elements of Programming Style*》出版至今已经有大约二十年的时间，书中所给出的一组规则到现在仍然可以作为最好的编程指导思想。然而目前，程序正在变得日益庞大和复杂，同时我们所使用的编程语言也已经发生了很大的变化。现在，我们不仅需要关注程序中每个组件的算法和数据结构，还需要关注如何将程序中的所有组件更好地组合在一起。DeRemer 和 Kron 提出了两个术语："大规模编程"（*programming-in-the-large*）和"小规模编程"（*programming-in-the-small*），这两个术语分别用来描述程序的"大规模"特性和"小规模"特性。"小规模编程"主要研究程序中"代码只有几页长"的组件，例如一个 C++类。而"大规模编程"则是研究如何将"小规模"的组件组合成一个完整的程序——用 C++的术语来说，就是研究类之间的关系。在 Kernighan 和 Plauger 的书中，重点讲解了"小规模编程"，而对于"大规模编程"的讨论虽然很有道理，但篇幅有限，基本上可以概括为以下两点：

1）将程序模块化。

2）有效地使用子程序。

本书所讲述的编程风格侧重于"大规模编程"，不过仅限于 C++编程领域。本书所面向的读者，是那些已经学习过 C++语言，并正在努力将语言的各种特性（尤其是面向对象的特性）应用于解决编程难题的程序员。虽然书中的讨论仅限于 C++，但对于其他语言而言，其中的许多编程经验同样适用。我在书中已经尽可能地保持了语言的独立性，以便于读者进行阅读。

我在本书中采用的是 Kernighan 和 Plauger 的讲解方法，即从对程序的关键的分析和改写中提炼出编程风格的规则。书中使用的所有程序都是从关于 C++编程的教科书、期刊和手册中节选出来的，我并没有专门为本书编写示例程序。其中有些程序是原封不动地照搬过来，而有些程序则是进行了修改。这些修改包括改正一些无伤大雅的小错误，以及在保留程序结构的基础上，对那些没有获得版权的程序进行的修改。

本书是本着"就事论事"的态度来分析代码的。我们都是通过阅读和分析彼此的代码来达到学习的目的。本书并不是要对某个程序员提出批评，只是试图找出好的程序与坏的程序之间的差别。毫无疑问，本书中的"好"程序也可能有着其自身的缺陷。我们鼓励读者对这些程序进行更严格的分析，并找出进一步改进编程风格的方法。

致谢

本书是在C++ At Work和各种USENIX会议所提供的资料基础上逐渐形成的。感谢SIGS的Rick Friedman以及USENIX的John Donnelly和Dan Klein，是他们为我提供了这些资料，同时也感谢编写这些资料的人们。特别感谢Solbourne公司的程序员们，他们为了给我提供第一份资料而殚精竭虑。本书还得益于《The C++ Journal》中的文章，因此我要感谢此期刊的编辑Livleen Singh。我与Dave Taenzer曾有过大量的讨论，这些讨论对于本书中的许多主题都有影响。同样还要感谢Addison-Wesley出版社的John Wait对本书的信心和耐心。

David Cheriton、James Coggins、Cay Horstmann、David Jordan、Brian Kernighan、Doug Lea、Scott Meyers、Rob Murray、Kathy Stark和Mike Vilot都审阅了本书的初稿，并提出了许多提高本书质量和消除错误的建议。我衷心地感谢他们的工作。

感谢我挚爱的Carol Meier，在我编写本书的时候，她自始至终都支持着我。在本书的构思和写作期间，我们迎来了我们的第一个孩子。

最后，我还要感谢以下的作者和出版机构，感谢他们允许我引用以下出版物中的资料：

Davis，S. R. 1991. *Hands-On Turbo C++*. Reading，MA：Addison-Wesley.

Dewhurst，S. C.，和Stark，K. T. 1989. *Programming in C++*. *Englewood Cliffs*，NJ：Prentice Hall.

C++ for *C Programmers* by Ira Pohl（Redwood City，CA：Benjamin/Cummings Publishing Company，1989）p. 92.

Stroustrup，B. 1991. The *C++Programming Language*，2d ed. Reading，MA：Addison-Wesley.

Wiener，R. S.，和Pinson，L. J. 1988. *An Introduction to Object-Oriented Programming and C++*. Reading，MA：Addison-Wesley.

Wiener，R. S.，和Pinson，L. J. 1990，*The C++Workbook*. Reading，MA：Addison-Wesley.

Shapiro，J. S. *A C++Toolkit*。本书的部分内容来源于*A C++Toolkit*，Shapiro，J. S，1991年版权所有。对这本书中部分内容的使用已经得到了许可。A C++Toolkit一书由Prentice Hall公司出版。

参考文献

Kernighan和Plauger［2］的书籍对所有程序员来说都是值得推荐的。DeRemer和

Kron [1] 提出了“大规模编程”和“小规模编程”这两个术语。

1. DeRemer, F., and Kron, H.“Programming-in-the-large versus Programming-in-the-samll”, *Proceedings of the International Conference on Reliable Software*, ACM/IEEE, April 1975, Los Angeles, California。

2. Kernighan, B. W., and Plauger, P. J. 1974 (2d ed., 1978). *The Elements of Programming Style*. New York, NY: McGraw-Hill。

Tom Cargill
波尔得，科罗拉多州

目　录

概　述

本书采用统一的形式来讲解。每章开始通过研究示例程序——“编程风格示例”——来引入每个学习主题，这些示例程序在某些重要的方面存在着缺陷。在分析程序时，我们的思路与进行代码交叉审查时的思路是一样的：在对同事的代码进行审查时，我们需要找出哪些问题是最需要改正的，以及对程序的哪些部分进行修改才能最大程度地提升程序的整体性能。在本书中，我们将对每个示例程序都进行详尽地阅读和分析。读者在阅读书中对示例程序的讲解之前，可以首先从自己的角度去分析程序中的问题，然后试着给出自己的解决方案。在完成了对示例程序的分析之后，我们还会把最初的程序和修改后的程序进行比较；在某些示例程序中，随着我们对程序的逐步改进，程序的代码量和复杂性都会显著地降低。最后，我们还可以对修改后的最终程序进行更严格的分析，并努力找出更多的改进方法。

对于从分析代码中所学到的每个主要知识点，我们都可以用一条规则来进行描述。所有这些规则在内容和侧重点上都有着很大的区别。其中有一些规则是具体的、技术上的规则，它们告诉程序员如何避开C++中常见的编程陷阱。而有些规则是关于如何在面向对象程序设计中构建抽象以及抽象之间的关系的。所有这些规则都与C++程序设计中的某些特性有着密切的关系。

我们需要重点理解的是在每条规则后面的动机和背景，而不是盲目地坚持这些规则。在某些情况下，程序员可以为了达到一个更重要的目标而去打破这些规则。事实上，本书中的某些规则之间是相互冲突的。例如，有的规则强调程序的简单性，而有的规则强调程序的完整性，这些规则通常都是相互冲突的。程序员不仅需要知道这些规则，而且还要理解其中的具体含义和使用环境，并且能够判断在什么时候应该打破这些规则。

在本书给出的所有示例程序中，有着许多不同的地方。下面对程序中各种不同的编写方式进行解释。

C++是一种特殊的编程语言，因为在许多不同层次的抽象上，我们都可以用C++来进行编程。例如，我们可以通过C++来对计算机硬件设备进行详细地控制，这基本上是用汇编语言才能达到的低层次抽象；此外，C++支持类、继承和多态，而这又是一种非常高层次的抽象。因此，在所有的面向对象语言中，从程序员所能控制的抽象层次上来看，C++是一种非常优秀的语言。

C++对C进行了许多扩展，其中的大多数扩展都是和大规模编程相关的。在C++中

引入了成员函数、访问控制、重载、继承和多态等特性，这些特性都是用来描述程序中不同组件之间的关系，或者一组程序之间的关系，而并不是用来描述如何在单个组件中进行编码。因此，在本书的C++编程风格中，我们所讨论的大部分问题都是关于如何描述组件之间的关系。事实上，很难在编程风格和软件设计之间划分出明显的界限。这是很自然的，因为与像C这样的语言相比，用C++来编程能够更明显地反映出软件的设计思想。我们通常是通过编程语言来表达具体的软件设计思想，而C++程序总是能够告诉我们大量设计细节。

本书假定读者已经熟练使用C++语言，这样我们就只需要解释少量的细节问题。读者可以通过自己所喜欢的入门书籍来学习那些尚不熟悉的语法和语义。只有当我们遇到一些深奥的技术细节时，才会进行明确地解释。对于某些C++的细节问题，我们建议读者去参考Ellis和Stroustrup的著作，或者翻阅 *The Annotated C++ Reference Manual*，Addison-Wesley，1990。一般很难学完Ellis和Stroustrup的参考手册式著作中所有内容，因此应该学会针对特定的问题有选择性地去阅读。

词法风格

在C++中可以使用许多不同的词法风格，例如缩进约定，在什么地方使用空格等。在本书中，我并没有指出哪一种词法风格是更好的。大多数的程序都是遵循着最初的词法风格。例如，在一些程序中将指向字符的指针声明为 `char *p`，而在有些程序中则使用的是 `char *p`。有时候我们用0来表示空指针，而有时候则使用NULL来表示。不过，在每个示例程序中，我们都将只用一种词法规则，并且在修改代码时也遵循这个规则。

我们并没有指出哪一种词法风格是更好的，理由有两个。首先，虽然在程序的可读性上，词法风格是很重要的，但与大规模编程的问题相比，这种重要性就显得苍白无力。例如，与程序中是需要一个类还是两个类，或者这些类是否通过继承来进行关联相比，我们在类的开始部分到底是声明公有成员函数还是私有成员函数就显得不那么重要。事实上，从某种意义上说，词法风格也可以是无足轻重的，我们可以通过一些自动工具——其中包括多种复杂程度的漂亮打印程序——将任意程序从一种词法风格转换到另一种词法风格。因此，在本书中将不会对词法风格进行讨论。

其次，许多关于词法风格的问题都是主观性的。例如，对某些程序员来说，多余的括号可以带来帮助，而对其他的程序员来说，多余的括号则妨碍了程序的编写。有些程序员喜欢看到返回表达式被写为 `return(x)`，而有些程序员则更喜欢使用 `return x`。对于不同的程序员来说，所有这些做法都是基于同一个理由："这样写更易于阅读"。程序员，无论是个人还是处于某个团队，必须自己决定所使用的词法风格。换句话说，即使我在词法风格上给出了我自己的观点，程序员也可以选择忽略它们。或许，唯一一条客观的原则就是：我们在软件项目的整个开发过程都应该只采用一种词法风格，并且应该自始至终都遵循这种词法风格。

内联函数与const声明

在示例程序中，有些成员函数被声明为内联函数，虽然这样的声明看上去对程序整体性能的提高并没有什么帮助。一些程序员只是例行公事地将代码量较小的成员函数声明为内联函数：将这些成员函数的定义放在类的声明中，这样就可以减少一些源代码的大小。然而，这通常是一种误解。在编译的时候，内联函数将在被调用的地方进行展开，这通常比相应的函数调用要占据更多的代码空间，尤其是当我们在内联函数中又调用了其他内联函数时。由于内联函数的展开所增加的代码量甚至可能降低程序的执行速度，因为代码量的增加将有可能妨碍有效的代码缓冲操作。只有当内联函数确实能够带来程序性能的提升时，才应该被用在代码中。然而，对于那些无法判断是否会带来性能提升的内联成员函数，我们还是应该将它们保留在最初的代码中，并且不予以分析。通常，在每个示例程序中，我们都会有一个关于正确性或者一致性的问题需要讨论，这个问题远比任何关于程序性能的问题重要。因此，我们将内联函数的相关问题就交给读者来分析。

在本书的示例程序中，对于const修饰符在类型和成员函数上的用法也是不同的。如果我们在程序中小心地使用const，那么就能够在编译期检测出一些错误。不过，由于在使用const的时候存在不同的风格，而且在一个已经完成的程序中加入const是非常困难的，因此，我们并没有将const修饰符增加到程序中去。const的出现或缺少并不会影响程序的整体结构，而整体结构才是我们关心的主要问题。

虚函数

虚函数的动态绑定是C++中一个功能强大并且重要的部分。本书中的大部分内容都是讨论在什么情况下应该将虚函数从程序中删除，而不是讨论在什么情况下虚函数应该被增加到程序中。我们将首先给出一些使用了虚函数的C++程序，然后对这些程序进行观察和分析，最后将重点放在如何消除虚函数上。使用虚函数的示例程序非常多，然而能够通过虚函数来改进性能的程序却非常少。因此，概括起来就是：在示例程序中，我们将重点说明如何避免虚函数，而不是说明如何使用虚函数。

注释

在本书中，大多数程序的注释都是很少的。事实上，从典型的软件产品的设计标准来看，这些注释是不够的。如果有更多的注释，那么读者在阅读程序时可能会更快，但这却违背了我们研究代码的初衷。如果注释不是很多，那么读者就必须直接从代码中获得更多的信息，从而把更多的注意力放在程序的细节上。通过对程序细节的分析来推断程序的设计思想，这对于理解大多数的示例程序来说都是非常重要的，这也是一个值得我们去努力培养的技术。

标准 I/O 与流

在本书的一些程序中，我们使用了标准的 C 语言 I/O 库（例如 printf）来对字符进行格式化输出，而在其他的程序中则是使用了流类库（例如 cout）。在大多数的情况下，无论程序使用的是哪种方式，都可以编写出同样易读和清晰的代码。

参数化类型和异常

在编写本书的时候，参数化类型（模板）在已发布的 C++编译器中是可以使用的，而异常处理则还是不支持的[⊖]。对于这些 C++特性来说，我们应该持谨慎的态度。我们应该等到大多数程序员都接受了这些特性，然后再去观察和分析使用了这些特性的程序，而不是凭主观想像去推测这些特性可能带来的混乱。只有当这些特性在程序中出现以后，我们才可以给出一些有意义的指导原则，这些原则用来告诉程序员在使用这些特性进行编程时，如何去避开常见的陷阱。

练习

在大多数章节的后面都给出了练习，其中一些练习是关于进一步阅读和分析程序。在一些练习的代码中，所存在的缺陷与该章示例程序中的缺陷是相同的。在其他的练习中，则给出了一些写得不错的程序。不过，这些练习并不是要求读者按照所给出的方式来编写代码。读者应该试着从零开始编写这些代码，而不是盲目地去遵循书中所给出的方式。

程序清单

在本书中，所有的示例程序都是从实际的源文件中节选出来的，这些源文件至少能够在一种 C++平台上进行编译和运行。其中大多数的程序都是在 Cfront 2.0 下运行的，还有一些是在 G++和 Borland C++ 3.0 下运行的。

参考文献

在编写本书的时候，对 C++的定义描述参考 Ellis 和 Stroustrup 的著作 [1]。

[1] Ellis, M. A., and Stroustrup, B. 1990. *The Annotated C++ Reference Manual.* Reading, MA: Addison-Wesley.

⊖ 这是因为作者编写本书的时候较早，这些特性在目前所有主流的 C++编译器上都是完全支持的。——译者注

抽　　象

在软件开发中，抽象处于中心地位，而类则是C++中最重要的抽象机制。类描述的是所有由这个类实例化的对象的共同属性，并且刻画了这些实例化的对象的共同行为。在C++程序设计中，正确地识别抽象是一个很关键的步骤。如果希望获得高质量的抽象，那么程序员就必须要充分地理解在程序中进行处理的对象的内在属性。

为了研究类这种抽象机制，我们首先来看一个示例程序，并评价这个程序的优缺点，特别要注意在程序中是如何来设计类的。然后，我们用不同的类与类之间的关系来改写程序。通过对程序结构的重新思考以及对示例程序的改写，我们就可以得到关于编程风格的普遍规则，这些规则对于改进其他的程序来说同样是适用的。

1.1　编程风格示例：计算机的定价

我们来观察程序清单1-1中程序所包含的类。在程序中包含的抽象是计算机、计算机组件以及它们的价格和折扣。程序的功能是确定一台计算机中各种配置的价格。在配置一台计算机时，我们需要作出两个选择：一个是扩展卡的选择，另一个是显示器的选择。扩展卡的插槽必须是以下三个选项之一：光驱、磁带机或者网络接口；显示器则必须是单色或者彩色的。我们在阅读程序的时候，要同时考虑如何对其进行简化。程序中的类是不是给出了正确的抽象？有没有其他方法能够通过改变程序中的抽象来降低程序的复杂性？

程序清单1-1　最初的程序

```
#include <stdio.h>

enum CARD { CDROM, TAPE, NETWORK };
enum MONITOR { MONO, COLOR };

class Card {
public:
virtual int  price() = 0;
virtual char *name() = 0;
virtual int  rebate();
};

class Network : public Card {
public:
    int  price();
    char *name();
};
```

```
class CDRom : public Card {
public:
    int  price();
    char *name();
    int  rebate();
};

class Tape : public Card {
public:
    int  price();
    char *name();
};

class Monitor {
public:
virtual int  price() = 0;
virtual char *name() = 0;
};

class Color : public Monitor {
public:
    int  price();
    char *name();
};

class Monochrome : public Monitor {
public:
    int  price();
    char *name();
};
int Card::rebate()    { return 45; }

int Network::price()  { return 600;       }
char *Network::name() { return "Network"; }

int CDRom::price()    { return 1500;    }
char *CDRom::name()   { return "CDRom"; }
int CDRom::rebate()   { return 135;     }

int Tape::price()     { return 1000;   }
char *Tape::name()    { return "Tape"; }

int Color::price()    { return 1500;    }
char *Color::name()   { return "Color"; }

int Monochrome::price()  { return 500;    }
char *Monochrome::name() { return "Mono"; }

class Computer {
    Card    *card;
    Monitor *mon;
public:
            Computer(CARD, MONITOR);
            ~Computer();
    int     netPrice();
    void    print();
};

int Computer::netPrice()
{
    return mon->price() + card->price()-card->rebate();
}
```

```
Computer::Computer(CARD c, MONITOR m)
{
    switch( c ){
    case NETWORK:   card = new Network; break;
    case CDROM:     card = new CDRom;   break;
    case TAPE:      card = new Tape;    break;
    }
    switch( m ){
    case MONO:      mon = new Monochrome; break;
    case COLOR:     mon = new Color;      break;
    }
}
Computer::~Computer()
{
    delete card;
    delete mon;
}

void Computer::print()
{
    printf("%s %s, net price = %d\n",
        card->name(), mon->name(), netPrice());
}

int main()
{
    Computer mn(NETWORK, MONO);
    Computer mc(CDROM,   MONO);
    Computer mt(TAPE,    MONO);
    Computer cn(NETWORK, COLOR);
    Computer cc(CDROM,   COLOR);
    Computer ct(TAPE,    COLOR);

    mn.print();
    mc.print();
    mt.print();
    cn.print();
    cc.print();
    ct.print();

    return 0;
}
```

程序清单 1-1 的程序输出如下：

```
Network Mono, net price = 1055
CDRom Mono, net price = 1865
Tape Mono, net price = 1455
Network Color, net price = 2055
CDRom Color, net price = 2865
Tape Color, net price = 2455
```

在 Computer 类的构造函数中，每个参数都是一个相应的枚举常量：

```
enum CARD { CDROM, TAPE, NETWORK };
enum MONITOR { MONO, COLOR };
```