国外计算机科学经典教材

Object-Oriented Design in C++
Using the Standard Template Library

C++ 面向对象程序设计

—— 使用 STL 编程

Nicholas J. De Lillo 著
袁勤勇 莫青 等译

清华大学出版社
http://www.tup.com.cn

C++面向对象程序设计

——使用 STL 编程

Nicholas J.De Lillo 著

袁勤勇 莫 青 等译

清华大学出版社

(京) 新登字 158 号
北京市版权局著作权合同登记号：01-2002-3037

内 容 简 介

本书主要介绍了面向对象设计的基本概念和思想，并通过大量实例深入地讲解了如何使用标准库进行面向对象程序设计。本书首先介绍了面向对象的一些基本概念，接着研究了标准模板库(STL)的主要组件，最后探讨了预定义容器类和泛型算法方面的知识。本书的每章内容中都包含了大量实用的练习，可以使读者很快地投入到面向对象设计的环境中。

本书适用于希望学习 C++面向对象程序设计的初学者，并可供使用标准库组件进行面向对象程序设计的开发人员参考。

图书在版编目(CIP)数据

C++面向对象程序设计——使用 STL 编程/(美)德李洛著；袁勤勇等译.—北京：清华大学出版社，2002
书名原文：Object-Oriented Design in C++ ——Using the Standard Template Library
ISBN 7-302-06007-X

Ⅰ.C… Ⅱ.①德…②袁… Ⅲ.C 语言—程序设计 Ⅳ.TP393

中国版本图书馆 CIP 数据核字(2002)第 081655 号
《C++》

出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http://www.tup.com.cn
责任编辑：胡辰浩
印 刷 者：北京通县大中印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　**印张**：24.5　**字数**：627 千字
版　　次：2002 年 11 月第 1 版　2002 年 11 月第 1 次印刷
书　　号：ISBN 7-302-06007-X/TP・3584
印　　数：0001～5000
定　　价：46.00 元

前　言

本书是为计算机科学或计算机工程课程编写的一本教材，可满足不同层次读者的需求。首先，本书可作为学习使用C++编程语言实现面向对象设计(OOD)的参考书。在第1、2章中，讲述了它的三个主要概念：封装、继承和多态性，并使用简单易懂的基本数学概念对它们进行了定义和举例说明。其次，本书介绍了标准模板库(Standard Template Library，STL)中的主要组件。第3、4两章先介绍了一些传统的概念，如：搜索、排序和散列。这两章为随后的几章奠定了基础，在后几章中介绍了包括标准模板库组件的预定义容器类和泛型算法。

本书前4章可作为数据结构和算法传统课程的基础内容，常被计算机协会(Association for Computing Machinery，ACM)作为CS7课程的内容。本书也可满足以下需求：

- 重点讲述标准库组件及其功能的计算机科学和计算机工程的高年级本科生和低年级研究生课程。
- 使用标准库(Standard Library)进行面向对象设计的计算机科学和计算机工程的高级本科生和低年级研究生课程。

本书不是为数据结构初级课程编写的(即ACM课程中的CS2课程)，因为已经有许多书籍可以满足这一课程的需求。本书主要介绍了使用标准库进行面向对象设计，并用大量的例子解释了数据结构课程中首先介绍的内容，如：链表、栈、队列和优先队列。然而，在CS2课程中，C++中这些数据结构的实现通常没有用到标准库。此外，本书中没有涉及CS2课程中讲述的树这一专题，因为STL的设计没有把树包含在容器类模板中。事实上，STL中用到树的地方可由使用红-黑树的排序关联容器类的实现来代替。

STL的主要观点是：它的使用不鼓励或强调面向对象设计的主要思想，特别是涉及到继承和多态性时。在9.6节和附录中，这些概念以及虚函数和类层次的概念与不同STL类模板中定义的组件的使用结合在一起。而这些类模板来自于两个独立的应用程序域。它的一个实现是图抽象数据类型，另一个实现是局域网的面向对象设计。

在用C++作为实现语言的数据结构和算法(ACM的CS7)课程的讲授中，一些新方法的关键就是使用标准库。这是由于标准模板库组件应用了商用软件设计中的另一个重要的目的，即强调了软件问题的正确性、效率、复用性和移植性。这些是软件工程课程中必讲的重要概念。本书中的程序代码设计也应用了这些概念。

目前，没有哪一本书给出了本书中的全部专题，这就是本书的特别之处。虽然有些书也介绍了本书中的某些内容，但是除了本书之外，没有哪本书包含了本书提到的全部内容。在许多书中，在介绍这些主要概念前并不提及预定义的标准模板库组件。然后在讲述标准模板库时又再次讲述这个专题，以保证正确性、效率和可移植性。与STL建立了某些关联后就达到了以下目的：

- 程序员不再需要考虑自己设计代码的正确性和效率，因为被访问的STL组件保证了正

确性和效率。

- 因为 STL 的设计与任意特定的硬件配置无关，所以其代码可用于所有平台。
- 使用 C++作为实现语言，并以软件开发专业人员为职业的学生获得了两个重要的里程碑。第一个是逐渐熟悉基本数据结构的理论描述；第二个是结构的实现细节(使用标准库)。

本书中，每一章的开始都列出了通过学习本章里讲述的概念来达到的目的。此外，每章的最后一节总结了该章讲授的最重要问题，以便于学生在忘记重要的概念时能找到相应的章节。每章最后的练习试图使学生使用该章描述的一些概念给出每个习题的解答，证明自己解决问题的能力，从过去的旁观者转变为实际的参与者。最后，每章的结束给出了至少一个程序设计项目，其难度超过了解决练习题所需的解题能力的一般水平。这些项目是学生感兴趣的一般理论的扩展，或是该章所描述的实现方法的另一种形式。每章中的习题和项目都经过了测试，大部分标准答案都可在附带的光盘中找到。

本书假定学生已经熟悉了数据结构的初级课程，因而书中所讲述的方法更为高级。对某些数据结构属性的描述只在需要加强本文可读性时才会提及。

本书是一本有用的参考书，它强调了面向对象程序设计方法的概念以及如何使用标准模板库组件来实现这些概念。书中介绍的概念大部分来自于软件设计中经常遇到的问题。以程序的形式给出它们的解答，其中有很多用到了标准模板库中的组件。每章中的练习都包括对该章所述概念的变化和概括。这些练习的范围包括从最基本的概念到对这些概念的挑战性的扩展。实际上，最具挑战性的练习是在独立的程序设计项目部分中。如前所述，设计这些练习的目的是使学生实际参与到面向对象设计的环境中，而不是只作为临时的看客。

以下列出了每一章所讲述的主题：

第 1 章：类。本章重述了在类的定义、设计和编码中出现的概念。讲述了构造函数、析构函数、数据成员和成员函数的概念，并用大量的例子说明了这些概念。强调了设计类的注意事项：被构造的对象的哪个属性应指定为私有(private)，哪个属性应指定为公有(public)。强调了数据抽象的概念和抽象数据类型(如 C++的类)的实现。本章还讨论了函数和类模板的概念、友元函数的使用、void*指针、异常和异常处理以及类的静态成员。通过大量的例子，使学生复习以前的课程中学习过的部分熟悉的概念。

第 2 章：继承和多态性。本章介绍并举例说明了某些概念的重要性，这些概念包括：类的层次关系、基类、派生类、抽象类、虚函数、类的保护成员以及静态和动态多态性。明确地强调了类的层次关系和多态性的价值及其在现代商用软件设计中的作用。

第 3 章：搜索和排序。本章首次用到了算法的概念，并给出了在排序和搜索中常用的算法。描述了这些算法的实现，而没有使用标准模板库中提供的程序，因为本章内容只是作为在第 8 章中提到的概念的预览。特别地，用 Big-O 分析方法定义了效率的概念及其量化；用有限归纳法定义了数学证明的概念。给出了递归编程方法和分而治之原则的思想。举例说明了在排序方法的传统问题中用到的概念。这些排序方法包括线性和二叉搜索、选择排序、插入排序、快速排序、合并排序以及其他排序方法。

第 4 章：散列：标准模板库的序幕。本章讨论了有关数据存储和在某种深度上进行检索的介绍性的专题。这些概念包括散列、散列函数、散列表、公开寻址、使用各种形式的探查方法

解决散列冲突以及链表等内容。此外，本章末尾讨论了命名空间，它提供了一种普遍的进入标准模板库环境的方法。

第 5 章：STL 中的组件概述。本章意欲在前 4 章中讲述的面向对象设计的传统特点和此后第 6 到第 9 章中讲述的使用 STL 组件的特点之间架起一座桥梁。概述了容器、泛型算法、迭代器、适配器和分配器的概念。从第 6 章开始将详述并举例说明这些概念。

第 6 章：队列容器。本章讲述了向量、双端队列和列表队列容器类模板的基础知识，详细介绍了这些概念在商用面向对象设计中的价值，特别强调了它们的使用效率。给出了使用这些形式的容器类模板的大量示例，并重点强调了如何选择合适的容器形式来有效地解决问题。

第 7 章：容器适配器。在第 6 章中介绍了基本队列容器后，本章介绍了容器适配器：栈、队列和优先队列，并把计算机科学中遇到过的问题作为例子来说明各个概念。这些例子主要来自于编译系统和操作系统的设计。内容包括检查圆括号、方括号和大括号是否成对出现，把算术表达式的前缀形式转变为等价的后缀形式，后缀算术表达式的赋值以及模拟多道程序时间共享操作系统中进程的就绪队列。

第 8 章：泛型算法。本章介绍了 STL <algorithm>，强调了库中大量函数的重要性。这些函数包括 lower_bound、binary_search、find、sort 等。本章并没有透彻全面地对这些内容进行解释，而是强调了在数据处理(例如复制、搜索和排序)中常使用的函数的效率。这些概念的大多数在第 3 章中已介绍过，本章将用<algorithm>中提供的强大而有效的函数来再次讲述这些概念。

第 9 章：排序关联容器。本章重点介绍了集、多集、映射和多重映射类模板，如何区分这些模板，并说明了它们在大量重要应用程序域中的应用。这些应用程序域包括排序、搜索和各种形式的图数据类型的定义和实现。在本章中，也将介绍关联数组这一重要概念以及如何在数据检索和散列中使用它。

附录：局域网模拟器。这里把 STL 中的概念的应用延伸到了一个重要的应用程序域：模拟局域令牌环网络的行为。

本书使用的全部代码已经在不同平台上进行了测试，包括 Visual C++ 6.0 版本和 Borland C++ 5.02 版本的编译器以及大量基于 UNIX 的平台。

目　　录

第1章　类

本章目的

- 讲述面向对象设计的重要概念，如：抽象、封装和模块化。
- 给出并解释了类和对象的定义及其在现代软件设计中的作用。
- 定义并解释了函数和类模板的概念以及它们在软件复用中的作用。
- 描述友元函数的概念及其应用。

1.1　简介

商用软件现代设计通常包括面向对象的编程风范。这使得解决软件问题的设计应该包含以下两个基本的目的：

- 正确性
- 效率

如果对定义了应用程序域的所有可能的输入，即定义应用程序或问题的输入值的集合，软件设计能提供预期的的解决方案，这样的设计才是正确的。换句话说，如果对其定义域内的所有值它都能正确地工作，此设计就是正确的。例如，在某一类型值的有限列表中搜索值的算法在以下情况下才是正确的：只要该值确实存在于列表中就能检索到它；若该值不在列表中就应给出如下的错误信息：

VALUE SOUGHT DOES NOT EXIST

只要被搜索的值不存在，就应给出提示信息。

此外，必须尽可能有效地设计软件的解决方案。这就意味着解决方案的执行应能相对快速地进行，并只使用其求解绝对必需的计算机资源。在许多情况下，解决方案的计算速度也许是产品成败的决定性因素。例如，在商用飞行器中使用的遥感勘测和引导系统，应能对天气情况的改变以及在飞行、降落和起飞时突然遇到的没有预期到的障碍尽可能快地作出反应。总之，为相对复杂的软件问题设计正确而高效的解决方案是需要努力的主要目标。以下将看到面向对象范例的正确实现通常都是满足这些目标的。

这些目标不是自动就能达到的，对设计和实现代码仍需做大量的准备和测试工作。C++设计语言有一优点，它拥有非常丰富和广泛的预定义的库的集合，这些库包含了在现代数据处理中遇到的许多算法的正确有效的程序编码。例如，标准模板库(STL)包含了解决大量基本问题所需的操作，如拷贝给定值的有限列表、以特有的顺序对值的有限列表进行排序、或在列表中搜索特有的值。

本书的主要目的是说明如何得到软件问题的正确而有效的解决方案。在学习了大量的不同

问题后，就能达到这一目的。这些问题的解决方案将考虑到这些目标，并使用 STL 中的组件，或是其他的 C++“高端”技术和概念。

1.2 面向对象设计的原则

在过去 30 年中，现代高级通用型设计语言的研究和发展注重于两个关键的因素：简单性和能力。就是说，致力于这样一种设计语言的发展：程序员只需编写相对简单的代码即可，这些代码容易被其他读者所理解，而且足够保证达到前述的正确性和效率的目标。

这一时期产生的许多语言都试图满足这些目标，但都在某个或多个方面失败了。有趣的是，可以注意到能够满足这些目标的解题方法大都实现了面向对象的设计思想。本书中所讲的 C++ 设计语言就是这种语言的一个例子。另一个是 Java，它在实现面向对象的设计思想上与 C++有许多不同之处。区别之一是，Java 是通过直接使用 Internet 来达到它的目的的。

面向对象设计的主要特征是：

- 抽象
- 封装
- 模块化

1.2.1 抽象

抽象是面向对象设计的重要特征。它是指语言用简单精确的术语实现理论概念的能力。通常，概念包括许多独立且相互作用的部分。把此概念应用到这一情形中，抽象使用户可以定义概念中不同的部分，并正常描述它们的功能。例如，在理论上把(值为类型 T 的)栈定义为空集或值的有序队列，在需要时能把值压入栈中(入栈)以及从栈中删除值(出栈)。值的出栈和入栈只能在一端，即栈顶进行。图 1-1 展示了栈这一抽象的概念。

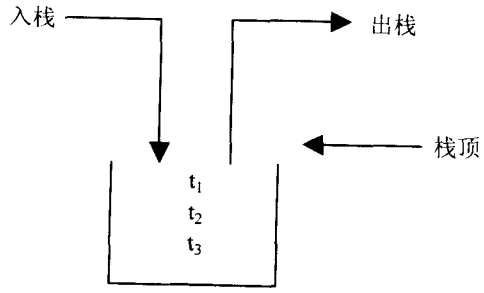

图 1-1

只要栈不为空，要测试当前栈是否为空或检索栈顶当前的值是很容易的。这给出了值为类型 T 的栈抽象的完整描述。当接受这一定义后，下一步就是使它适合于抽象数据类型(ADT)的形式。要定义 ADT 作为某一数据结构的数学模型，可通过以下方式来实现：指定此结构中存储的数据类型，以及允许对这些数据进行的操作和正确有效地实现这些操作所需的参数。应注意的是，ADT 描述了允许进行何种操作，但不必给出如何执行这些操作的任何信息。

例如类型 T 值的栈这一抽象数据结构包括两部分：

- 包含全部值为类型 T 的栈的应用程序域。
- 可应用于应用程序域内任一成员的允许操作的有限集合。

在栈这一例子中，列出了以下允许的操作：

- 入栈操作，把值压入当前栈中。
- 出栈操作，在当前栈非空时从栈中删除值。
- is-empty(为空)操作，检测当前栈是否为空。
- 栈顶操作，在当前栈非空时检索栈顶的值。

这些操作描述了类型 T 值的 ADT 栈的所有特征。在 C++中，此 ADT 一般作为类来实现，类定义了存储在数据结构中的数据类型和应用于类实例的允许操作。这些实例就称为该类的对象。

在 C++中，类的设计一般包含两个主要的部分：

- 用户接口
- 实现文件

在描述面向对象设计的其他特点时，这种细分是很有用的。

1.2.2 封装

面向对象设计的另一个重要的原则就是封装，或称为信息隐藏。这一原则首先由 David Parnas 在 1971 年提出。[1] 回顾如前所述的 ADT 类的实现，封装这一思想加强了面向对象设计的两个主要特征：

- 类必须且只需把实现抽象所需的全部信息提供给有意的客户(用户)。
- 类定义必须且只需为实现提供完成类定义所需的全部信息。

这样，类的使用者就无须了解其实现过程，也不必编写依赖于特有实现的程序。这使得类的维护变得简单，因为类的设计者(实现者)知道可以修改哪部分代码而不会影响到使用者的应用程序。从另一角度来看，一旦设计好类并提供给用户使用，除了在用户接口中提供的必需的功能外，实现者就没有办法知道客户使用类所做的全部具体应用。这使得用户程序的维护变得简单，因为程序员确切地知道哪部分代码可以修改，哪部分不可以修改。

在 ADT 栈这一示例中包含了 C++类定义。其用户接口的描述把用户可用的操作列在了接口的公有(public)部分，而把需要隐藏的实现细节列在了私有(private)部分。这样，用户接口将包含用户可用的栈操作的名字列表，例如 push、pop、is_empty 和 top，而在私有部分表示一些实现细节。在用户接口公有部分中描述的操作只需描述为原型，因为用户必须知道这些操作的具体名字和数量、输入值出现的顺序、它们各自的类型以及返回值的类型。用户没有必要了解这些操作是如何实现的，这部分是实现文件要考虑的内容。

例如，对于 ADT 栈，就有两种常用的实现方式，后边会详细地进行讨论。这两种实现为：

- 队列实现
- 链接实现

这两种方式分别给出了 4 种允许的操作(push、pop、is_empty 和 top)的实现。用户没有这

1 参看本章末尾参考文献中 parnas 的文章。

两种实现的直接接口，他们不必考虑实现的细节。实际上，在 ADT 栈的类定义设计中，后边的决定可能会将队列实现修改为链接实现。对于信息隐藏这一思想，这种改变不会影响用户的应用程序。

因此，就可以说栈的类定义封装了 ADT 栈的定义。因此，由封装得到了适应性，因为它允许修改部分程序的实现细节，而不会影响程序代码的其他部分。类定义中的公有和私有部分给出了如何把信息隐藏应用于面向对象的范例中。

1.2.3 模块化

现代商用软件系统典型地包含了许多有组织和有计划的自治单元，通常称它们为模块。模块的设计应能使模块正确地相互作用，以生成所设计的问题的解决方案。在面向对象系统环境中，这些模块的设计和适当的组织就称为模块化。它包括了大量类的设计，这些类的对象必须恰当地相互配合以生成想要得到的解决方案。在许多情况中，可能要用层次结构来设计类，即指某些类是基于现有的更一般的类来设计的。在面向对象设计中，这一实现运用了继承的概念。在继承关系中，定义的基类是由一个或多个所属的派生类组成的，派生类的对象继承了基类中的某些属性。

在类的继承结构中，由派生类构造的任何对象与相应的基类满足“is-a”关系。下面将举例说明这一概念。

例 1.1 假设有一组代表了二维几何图形简单层次关系的类，其关系如图 1-2 所示。

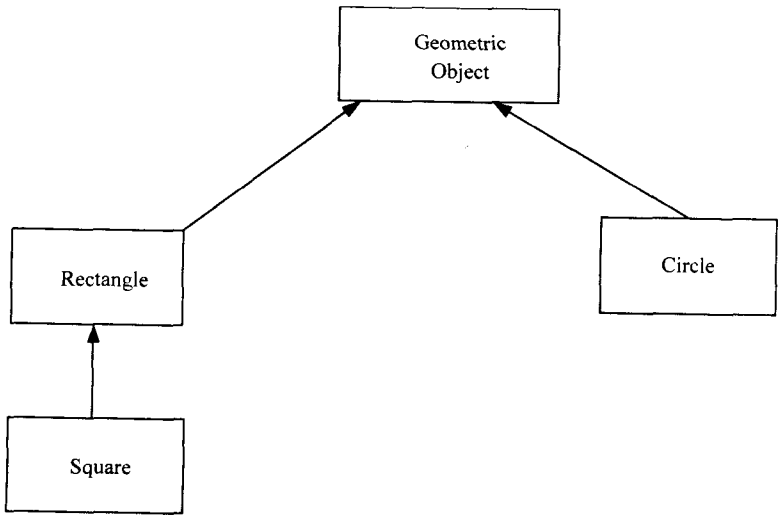

图 1-2

这一组织结构为：在每个盒子中都定义了一个单独的类，并有如图所示的从属关系。这样，由类 Rectangle 构造的对象是 Geometric object 对象的特例，这一从属关系也适合于由类 Circle 构造的对象。这一层次可下降一级，即构造 Square 对象，它既是 Rectangle 对象的特例，也是 Geometric object 对象的特例。

类的层次关系在设计大型软件问题的解决方案时通常是很有用的。这是因为有层次的模块化设计的实现代表了大量对象的共有功能，这些对象来自不同的带有普通的和更为一般的基类

功能的派生类。此外，由派生类构造的对象有各自不同的特殊行为。

使用模块化设计促进了软件的复用性。这意味着为解决特定软件问题而设计的模块也许能用来解决大量看似无关的软件问题。在这一情况中，就没有必要进行重新编码，只需把用于解决原来问题的模块插入或链接到新系统的模块中即可。这在C++中当然是可能的，为解决某一问题而设计的类具有可以被再次使用解决其他问题的功能，因为这一功能也能应用于其他问题。

1.3 类和对象

类无疑代表了C++中实现的面向对象设计的中心结构。类能把相关的数据项和对这些数据进行的操作组合到一起。这一集合包含着一组明确定义了的数据以及允许对那些数据进行的操作的有限集合，这些允许操作已作为 ADT 的必要成分定义好了。因此，定义了类后就能实现ADT这一概念。

在C++中，定义类的用户接口需要以下操作：首先用关键字class引导类声明，接着是命名类的标识符，类的内容(或体)被包含在花括号内，在公有部分描述(也许用原型的格式)类的成员函数，而在私有部分描述数据成员，类的用户接口的描述要以分号(;)结束。

定义类的用户接口的语法形式表示如下：

```
class class_name{
    class_body
};
```

其中，class 是 C++中的关键字，class_name(类名)是命名类的标识符，class_body(类体)包含着类成员(数据成员和成员函数)的定义。数据成员包含着定义类所需的数据，成员函数通常描述了允许对这些数据的进行的操作。

要管理用户对类成员的访问，可通过在类体中的定义前使用关键字 public 和 private (在某些使用继承的实例中也用到了 protected)。默认的访问权限为 private。指定为私有的类成员只能由该类成员访问。如果类的操作是公有的，则任何用户都可以使用这一操作。这样，若类的成员需要提供给任意用户使用，则其必须出现在函数体中并用关键字 public 作为声明的开始。

用类名来声明其数据类型的变量就是该类的对象。这就是说，它是名为类名的基础 ADT 的一个实例。类的对象的声明中要包含构造函数。构造函数可以为新对象的数据成员分配内存以便在需要时(取决于构造函数的形式)存放初始值，或为构造同一个类的现有对象的复制对象分配内存。构造函数很容易识别，因为它总是与类有相同的名字。实际上，类能有多个构造函数，也可以在它的定义中提供非显式的构造函数。下一节中将对这部分内容进行详细的介绍。

类体中可以包含指明该类对象行为的变量或常量定义。类体块中定义的变量和常量与普通的变量和常量有一个差别。这些变量或常量只在创建该类对象时才会被分配内存。实际上，如果在类体中定义了变量或常量，就只在构造该类对象时才会同时给它们分配内存。

类的出现是如何在现代软件设计的一般实践中发挥作用的呢？设计问题的软件解决方案一般要经过3个基本的阶段。首先，要以尽可能精确的方式得到对问题的清晰明了的理解。这称为软件开发的分析阶段。接下来是设计阶段，即识别包含在解决方案中的关键成分。最后是程