```
}

double cosDegrees(double angle) {
   return cos(toRadians(angle));
}

double toDegrees(double radians) {
   return radians * 180 / PI;
}

double toRadians(double degrees) {
   return degrees * PI / 180;
}
```

图 2-9 （续）

2.7 接口设计原则

程序设计的一个难点是程序需要考虑到底层应用的复杂性。随着计算机处理的问题越来越复杂，程序也会变得越来越复杂难懂。

编写一个处理大型或复杂问题的程序将迫使你面对越来越惊人的程序复杂性的增长。有许多算法需设计，许多特殊情况需处理，用户的需求需满足，以及大量细节问题需要兼顾。为了提高程序的可管理性，你必须尽你所能地降低程序的复杂性。函数可以降低一部分程序复杂度；类库也提供了同样可用于降低复杂性的方法，但同时也需要你在建库时考虑更多的细节。函数向调用者提供了能完成特定功能的一组操作集。类库向用户提供了一组函数和数据类型，以实现计算机科学家所描述的**编程抽象**（programming abstraction）。一个特定的抽
85 象能多大程度地简化程序取决于你可以多好地设计接口。

为了设计一个高效的接口，你必须平衡多个方面的需求。通常，你应该以下面的准则来尝试构建接口：

- 统一性（unified）。一个接口必须依照一个统一的主题来定义一致的抽象。如果某个函数不符合这一主题，就必须使用其他方法来重新定义这一函数。
- 简单性（simple）。库的底层实现是复杂的，但是库的接口必须向用户隐藏这种复杂性。
- 充分性（sufficient）。当用户使用一种抽象时，接口必须提供足够的功能来满足用户的需求。如果接口缺少一些关键功能，用户可能会拒绝使用并自己开发更好的库。与简单性同样重要，库的设计者必须避免在简单化过程中使得库的功能大大减少。
- 通用性（general）。一个良好设计的接口必须有高度的适用性，它可以满足不同用户的需求。一个只为特定用户提供的小范围操作集的库，其可用性将大大低于一个可以在多种情况下使用的库。
- 稳定性（stable）。接口中定义的函数在底层实现改变时，也必须拥有一丝不变的函数接口结构和函数功能。函数功能的改变将迫使用户向接口的变化妥协并修改程序。

下面，将详细讨论上述的接口设计准则。

2.7.1 统一主题的重要性

团结就是力量。

——伊索，《一捆柴枝》，～公元前 600 年

设计良好的接口所应具备的核心特征是该接口体现了一个统一的、一致的抽象。从

某种程度上说，这一库设计标准反映了库中选择的函数必须拥有一个一致的主题。因此，<cmath>库提供了数学函数，<iostream>库提供了cin、cout和cerr数据流以及执行输入输出操作符，2.6节介绍的error.h接口提供了一个报告错误的函数。这些库提供的每一个接口入口都符合接口的主题。例如，你不会让<string>库提供sqrt函数，因为sqrt函数更加符合<cmath>库的主题框架。

统一主题原理也对库接口中的函数设计产生影响。在一个接口中的函数应该尽可能在功能上表现出一致性。例如，<cmath>库中函数处理的角度值用弧度表示。如果有一些函数使用度数表示角度值，用户就不得不去记住每个函数要使用的参数的单位。 86

2.7.2 简单性与信息隐藏原理

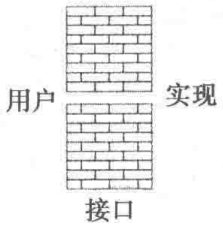
见素抱朴。

——老子，《老子》，～公元前550年

使用接口的首要目标就是降低编程的复杂性，因此很容易理解，简单性也是接口设计最迫切的目标。通常来说，一个接口应该尽可能降低使用难度。虽然底层实现的操作可能极度错综复杂，但是应该让用户可以从一个更简单、更抽象的角度去了解这些操作。

在某种程度上，接口扮演着一个特定库抽象参考指南的角色。当你想要知道怎样使用error函数的时候，你只需要查询error.h接口来了解使用方法。接口包含了你所需要的全部信息。对于用户来说，知道太多和太少的信息同样都是不利的，因为多余的细节信息将使界面变得更难以理解。通常，接口所含的数据信息应隐藏在接口中而不应对外暴露。

当你设计一个接口时，你应该尽可能地使用户远离其实现的复杂细节。从这个角度上讲，最好的办法是将接口设计成一堵分隔用户和实现的墙，而不是这两个部分间的通信渠道。



像希腊神话中分隔皮拉莫斯和提斯伯的墙一样，这堵墙在用户和实现之间提供了一道缝隙来使双方进行沟通。在编程时，这一缝隙包含了允许用户和实现之间进行沟通的界面入口。但是这堵墙的主要目标是保持用户和实现的分离。由于我们总是将它看成代表库的抽象的一道边界，所以接口有时也称作**抽象边界**（abstraction boundary）。理想情况下，库中所有复杂的部分都会被隔离在实现的一边。一个成功的接口将使用户远离那些复杂的实现。保持实现细节被限制在库的实现中又被称为**信息隐藏**（information hiding）。

信息隐藏原理在接口的设计上具有重要的现实意义。当你编写一个接口时，应该确保没有公开其实现的细节，包括注释部分。特别是当你同时编写接口和实现程序的时候，你可能会忍不住想要在接口中介绍你在实现库时用到的巧妙方法。试着放弃这种想法，接口的目标是使用户感到方便，应该只包含用户需要知道的信息。 87

同样，你在接口中应该尽可能将函数设计得足够简单。如果你可以减少函数参数的数量，或者有减少函数发生特殊情况的方法，都将使函数变得更加简单易用。除此之外，限制一个接口提供的函数数量也是有利的，这样做可以使得用户不会在大量的函数中迷失自己而

缺乏对整个接口的了解。

2.7.3 满足用户需求

每一件事都应尽可能的简单，但不能过度。

——阿尔伯特·爱因斯坦

简单易用只是设计库要求的一部分。为了使库变得简单，最容易的做法就是删除库中复杂和难懂的部分。但这么做也会使你设计的库的可用性降低。有时用户需要执行一些具有内在复杂性的任务。所以为了接口的简单而拒绝用户的需求不是一个好想法。为了更好地服务于用户，你的库必须提供充足的函数。学会在设计库的时候平衡好简单性和完整性是一项重要而艰巨的任务。

在许多情况下，接口的用户考虑的不只是库是否提供了某项功能，他们同时也在考虑这项功能的实现是否是高效的。例如，如果你正在开发一个空中交通管理系统，并且该系统需要使用库中的函数，你会要求这些函数必须快速返回正确的数值。过慢的应答与错误的返回值都将酿成灾难性的后果。

在大多数时候，程序执行效率更多的与库的实现有关，而与接口无关。即使这样，你也会经常发现在设计接口时考虑实现策略是很有意义的。假如让你在两种设计中做出选择，其中一种设计的实现更加简单高效，而另一种并没有让你有不得不选的其他原因，那毫无疑问你会选择简单高效的设计。

2.7.4 通用工具的优势

给我们工具，我们将完成任务。

——温斯顿·丘吉尔，1941 年 BBC 广播演讲

88 一个完全适合某种特定类型用户的接口，并不一定适合其他类型的用户。一个好的库抽象必须服务于多种类型用户的需求。为了达到此目的，我们必须在设计的时候提高库的通用性，以使得它可以解决多元化的问题而不是受限于某个非常特殊的应用情况。为了选择一种灵活性强的设计，你可以创建多用途的接口。

确保接口的通用性具有重要的现实意义。当你编写一个程序时，会经常发现你需要一个特殊的工具。如果你认为这个工具非常重要以至于需要把它放入到库中，那么你就需要转换你的思维模式了。当你为那个库设计接口的时候，你必须遗忘这个库最初的应用环境，而应为更通用的受众设计你的接口。

2.7.5 库稳定性的价值

人们改变了，却忘了彼此告知。太糟糕了——这导致了许多错误。

——莉莉安·海尔曼，《阁楼里的玩具》，1959

接口的另一特性使得它们在编程中具有关键的作用：它们可以在很长一段时间中保持自身的稳定。一个稳定的接口通过创建清晰的责任边界可以大大简化系统的维护。在库接口不变的情况下，实现者和用户都可以相对自由地改变自身负责部分的代码。

例如，想象一下你是 <cmath> 库的实现者。在你工作的时候，发现了一个实现 sqrt 函数的更精妙方法，这个方法可以使得计算平方根的时间缩小到一半。如果可以向用户告知你有一个 sqrt 函数的更好的实现方法，这个方法比以往更快，他们会非常高兴。但从另一

方面来说，如果你告诉用户函数的名称将改变或者说函数具有新的约束，那么用户会感到愤怒。为了使用你提供的“改进版”平方根函数，他们需要被迫修改自己的程序。修改程序是一项耗费时间，易于出错的活动，很多用户宁愿放弃程序的一部分性能提升，也不愿意他们的程序经修改后而无法运行。

接口只有在保持稳定的情况下才可简化程序的维护。当出现新算法或者应用需求变化时，程序经常被修改。然而在程序的这种进化中，其接口应尽可能地保持不变。在一个良好设计的系统中，修改实现是一个直截了当的过程。修改所涉及的复杂性只限定于抽象边界的实现部分。另外，修改接口通常会导致依赖于该接口的所有程序都必须进行修改。因此，修 89
改接口应是非常罕见的行为，这一行为应仅在用户参与时进行。

某些接口的修改造成的影响将会比其他接口的修改造成的影响更大。例如，在库中添加一个全新的函数是一个直截了当的工作，因为并没有用户的程序使用过这个函数。像这样接口修改使得以前使用该库的程序不用进行修改的行为称为接口的**扩展**（extending）。如果你发现在一个接口的生命周期中需要对接口进行一次升级，最好的办法是通过扩展来达到此目的。

2.8 随机数库的设计

领会接口设计原则最容易的方法就是进行一次简单的设计实践。为达到此目的，本节将讲述开发 Stanford 类库中 `random.h` 接口的整个设计过程，它便编写做出似随机选择的程序成为可能。模拟随机过程是必须的，例如，如果你想要编写抛硬币或者掷骰子的电脑游戏，你就需要模拟随机过程，当然，这一过程在实际应用中还有着更重要的作用。模拟随机事件的程序称作**不确定性的**（nondeterministic）程序。

让计算机表现出随机选择的行为是比较复杂的。为了给用户程序提供便利，你必须隐藏接口背后的复杂过程。本节你将有机会从接口设计者、接口实现者和用户这几个不同的角度来了解这个接口。

2.8.1 随机数与伪随机数

使用计算机生成随机过程的概念经常被描述为生成特定范围内的一个**随机数**（random number），部分原因是早期计算机主要用于处理数值应用。从理论上讲，一个无法预测其值，且在其取值范围内其值等概率出现的数被称为随机数。例如，掷骰子时出现一个范围从 1 到 6 的随机数。如果骰子是正常的，我们将无法准确预测掷出的数字。并且六个值出现的概率均等。

虽然随机数的概念看起来如此直观，但是要将这一概念在计算机上实现是有困难的。因为计算机总是执行内存中一系列特定序列的指令，因此计算机的函数都是遵循确定的模式的。怎么让遵循确定序列指令的计算机产生不可预测的结果呢？如果一个数是一个确定过程 90
的结果，那么，任何用户都应该能通过一组相同的指令来预测出计算机的响应。

事实上，计算机正是通过使用特定的过程来产生所谓的随机数。这一策略之所以可行，是因为理论上用户可以通过同样的一组指令预测出计算机的结果，但实际上并没有人会无聊到去做这种事。在大多数现实应用中，产生的数字是否真正是随机的关系并不大，真正有影响的是该数字看起来是不是随机的。为了使产生的数字看起来是随机的，它们必须具备如下特点：（1）从统计的观点来看，该数表现的像一个随机数；（2）事先要预测这个数的值应该很难，以至于没有人想去预测它。通过计算机中的一个特定算法来产生的“随机数”也被称