Addison Wesley

TURING

图灵程序设计丛书

C语言接口与实现

创建可重用软件的技术

英文版

C Interfaces and Implementations

Techniques for Creating Reusable Software

[美] David R. Hanson 著

人民邮电出版社
POSTS & TELECOM PRESS

TURING

图灵程序设计丛书

C语言接口与实现

创建可重用软件的技术

英文版

C Interfaces and Implementations

Techniques for Creating Reusable Software

[美] David R. Hanson 著

人民邮电出版社
北京

图书在版编目（CIP）数据

C语言接口与实现 : 创建可重用软件的技术 = C Interfaces and Implementations: Techniques for Creating Reusable Software : 英文 / (美) 汉森 (Hanson, D. R.) 著. -- 北京 : 人民邮电出版社, 2010.8
(图灵程序设计丛书)
ISBN 978-7-115-23113-0

Ⅰ. ①C… Ⅱ. ①汉… Ⅲ. ①C语言－接口设备－程序设计－英文 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2010)第096195号

内 容 提 要

本书概念清晰、实例详尽，是一本有关设计、实现和有效使用C语言库函数，掌握创建可重用C语言软件模块技术的参考指南。书中提供了大量实例，重在阐述如何用一种与语言无关的方法将接口设计实现独立出来，从而用一种基于接口的设计途径创建可重用的API。

本书是所有C语言程序员不可多得的好书，也是所有希望掌握可重用软件模块技术的人员的理想参考书，适合各层次的面向对象软件开发人员、系统分析员阅读。

图灵程序设计丛书

C语言接口与实现：创建可重用软件的技术（英文版）

◆ 著 [美] David R. Hanson
责任编辑 朱 巍

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：33.5
字数：629千字 2010年8月第1版
印数：1－2 500册 2010年8月北京第1次印刷

著作权合同登记号 图字：01-2010-4227号

ISBN 978-7-115-23113-0

定价：79.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

如今的程序员忙于应付大量关于 API（Application Programming Interface）的信息。但是，大多数程序员都会在其所写的几乎每一个应用程序中使用 API 并实现 API 的库，只有少数程序员会创建或发布新的能广泛应用的 API。事实上，程序员似乎更喜欢使用自己搞的东西，而不愿意查找能满足他们要求的程序库，这或许是因为写特定应用程序的代码要比设计可广泛使用的 API 容易。

不好意思，我也未能免俗：lcc（Chris Fraser 和我为 ANSI/ISO C 编写的编译器）就是从头开始编写的 API。（在 *A Retargetable C Compiler: Design and Implementation* 一书中有关于 lcc 的介绍。）编译器是这样一类应用程序：可以使用标准接口，并且能够创建在其他地方也可以使用的接口。这类程序还有内存管理、字符串和符号表以及链表操作等。但是 lcc 仅使用了很少的标准 C 库函数的例程，并且它的代码几乎都无法直接应用到其他应用程序中。

本书提倡的是一种基于接口及其实现的设计方法，并且通过对 24 个接口及其实现的描述详细演示了该方法。这些接口涉及很多计算机领域的知识，包括数据结构、算法、字符串处理和并发程序。这些实现并不是简单的玩具，而是为在产品级代码中使用而设计的。实现的代码是可免费提供的。

C 编程语言基本不支持基于接口的设计方法，而 C++ 和 Modula-3 这样的面向对象的语言则鼓励将接口与实现分离。基于接口的设计跟具体的语言无关，但是它要求程序员对像 C 一样的语言有更强的驾驭能力和更高的警惕性，因为这类语言很容易破坏带有隐含实现信息的接口，反之亦然。

然而，一旦掌握了基于接口的设计方法，就能够在服务于众多应用程序的通用接口基础上建立应用程序，从而加快开发速度。在一些 C++ 环境中的基础类库就体现了这种效果。增加对现有软件（接口实现库）的重用，能够降低初始开发成本，同时还能降低维护成本，因为应用程序的更多部分都建立在通用接口的实现之上，而这些实现无不经过了良好的测试。

本书中的24个接口引自几本参考书，并且针对本书特别做了修正。一些数据结构（抽象数据类型）中的接口源于lcc代码和20世纪70年代末到80年代初所做的Icon编程语言的实现代码（参见R.E.Griswold和M.T.Griswold所著的*The Icon Programming Language*）。其他的接口来自另外一些程序员的著作，我们将会在每一章的参考资料（Further Reading）部分给出详细信息。

书中提供的一些接口是针对数据结构的，但本书不是介绍数据结构的，本书的侧重点在算法工程（包装数据结构以供应用程序使用），而不在数据结构算法本身。然而，接口设计的好坏总是取决于数据结构和算法是否合适，因此，本书可算是传统数据结构和算法教材（如Robert Sedgewick所著的*Algorithms in C*）的有益补充。

大多数章节会只介绍一个接口及其实现，少数章节还会描述与其相关的接口。每一章的Interface部分将会单独给出一个明确而详细的接口描述。对于兴趣仅在于接口的程序员来说，这些内容就相当于一本参考手册。少数章节还会包含Example部分，会说明在一个简单的应用程序中接口的用法。

每章的Implementation部分将会详细地介绍本章接口的实现代码。有些例子会给出一个接口的多种实现方法，以展示基于接口设计的优点。这些内容对于修改或扩展一个接口或是设计一个相关的接口将大有裨益。许多练习题会进一步探究一些其他可行的设计与实现的方法。如果仅是为了理解如何使用接口，可以不用阅读Implementation一节。

接口、示例和实现都以literate程序的方式给出，换句话说，源代码及其解释是按照最适合理解代码的顺序交织出现的。代码可以自动地从本书的文本文件中抽取，并按C语言所规定的顺序组合起来。其他也用literate程序讲解C语言的书籍有*A Retargetable C Compiler*和D.E.Knuth写的*The Stanford GraphBase: A Platform for Combinatorial Computing*。

本书架构

本书材料可分成下面的几大类：

基础
1. 简介
2. 接口与实现
4. 异常与断言
5. 内存管理
6. 再谈内存管理

数据结构	7. 链表
	8. 表格
	9. 集合
	10. 动态数组
	11. 序列
	12. 环
	13. 位向量
字符串	3. 原子
	14. 格式化
	15. 低级字符串
	16. 高级字符串
算法	17. 扩展精度算法
	18. 任意精度算法
	19. 多精度算法
线程	20. 线程

建议大多数读者通读第 1 ～ 4 章的内容，因为这几章形成了本书其余部分的框架。对于剩下的 5 ～ 20 章，虽然某些章会参考其前面的内容，但影响不大，读者可以按任何顺序阅读。

第 1 章介绍了 literate 程序设计和编程风格与效率。第 2 章提出并描述了基于接口的设计方法，定义了相关的术语，并演示了两个简单的接口及其实现。第 3 章描述了 Atom 接口的实现原型，这是本书中最简单的具有产品质量的接口。第 4 章介绍了在每一个接口中都会用到的异常与断言。第 5 章和第 6 章描述了几乎所有的实现都会用到的内存管理接口。其余各章都分别描述了一个接口及其实现。

教学使用建议

我们假设本书的读者已经在大学介绍性的编程课程中了解了 C 语言，并且都实际了解了类似《C 算法》一书中给出的基本数据结构。在普林斯顿，本书是大学二年级学生到研究生一年级的系统编程课程的教材。许多接口使用的都是高级 C 语言编程技巧，比如说不透明的指针和指向指针的指针等，因此这些接口都是学习这些内容非常好的实例，对于系统编程和数据结构课程非常有用。

这本书可以以多种方式在课堂上使用，最简单的就是用在面向项目的课程中。例如，在编译原理课程中，学生通常需要为一个玩具语言编写一个编译器。在图形学课程中同样也经常有一些实际的项目。本书中许多接口消除了新建项目所需

要的一些令人厌烦的编程工作，从而简化了这类课程中的项目。这种用法可以帮助学生认识到在项目中重用代码可以节省大量劳动，并且引导学生在其项目中对自己所做的部分尝试使用基于接口的设计。后者在团队项目中特别有用，因为“现实世界”中的项目通常都是团队项目。

普林斯顿大学二年级系统编程课程的主要内容是接口与实现，其课外作业要求学生成为接口的用户、实现者和设计者。例如其中的一个作业是这样的，我给出了8.1节中描述的Table接口、它的实现的目标代码以及8.2节中描述的单词频率程序wf的说明，让学生只使用我们为Table设计的目标代码来实现wf。在下一个作业中，wf的目标代码就有了，他们必须实现Table。有时我会颠倒这些作业的顺序，但是这两种顺序对大部分学生来说都是很新颖的。他们不习惯在大部分程序中只使用目标代码，并且这些作业通常都是他们第一次接触到在接口和程序说明中使用半正式表示法。

最初布置的作业也介绍了作为接口说明必要组成部分的可检查的运行时错误和断言。同样，只有做过几次这样的作业之后，学生们才开始理解这些概念的意义。我禁止了突发性崩溃，即不是由断言错误的诊断所宣布的崩溃。运行崩溃的程序将被判为零分，这样做似乎过于苛刻，但是它能够引起学生们的注意，而且也能够让学生理解安全语言的好处，例如ML和Modula-3，在这些语言中，不会出现突发性崩溃。（这种评分方法实际上没有那么苛刻，因为在分成多个部分的作业中，只有产生冲突的那部分作业才会判为错误，而且不同的作业权重也不同。我给过许多0分，但是从来没有因此导致任何一个学生的课程总成绩降低达1分。）

一旦学生们有了自己的几个接口后，接下来就让他们设计新的接口并沿用以前的设计选择。例如，Andrew Appel最喜欢的一个作业是一个原始的测试程序。学生们以组为单位设计一个作业需要的任意算术精度的接口，作业的结果类似于第17章到第19章中描述的接口。不同的组设计的接口不同，完成后对这些接口进行比较，一个组对另一个组设计的接口进行评价，这样做很有启迪作用。Kai Li的那个需要一个学期来完成的项目也达到了同样的学习实践效果，该项目使用Tcl/Tk系统（参见J.K. Ousterhout所著的*Tcl and the Tk Toolkit*）以及学生们设计和实现的编辑程序专用的接口，构建了一个基于X的编辑程序。Tk本身就是一个很好的基于接口的设计。

在高级课程中，我通常把作业打包成接口，学生可以自行修改和改进，甚至改变作业的目标。给学生设置一个起点可以减少他们完成作业所需的时间，允许他们做一些实质性的修改鼓励了有创造性的学生去探索新的解决办法。通常，那些不成功的方法比成功的方法更让学生记忆深刻。学生不可避免地会走错路，为

此也付出了更多的开发时间。但只有当他们事后再回过头来看，才会了解所犯的错误，也才会知道设计一个好的接口虽然很困难，但是值得付出努力，而且到最后，他们几乎都会转到基于接口的设计上来。

如何得到代码

本书中的代码已经在以下平台上通过了测试：

处理器	操作系统	编译器
SPARC	SunOS 4.1	lcc 3.5 gcc 2.7.2
Alpha	OSF/1 3.2A	lcc 4.0 gcc 2.6.3 cc
MIPS R3000	IRIX 5.3	lcc 3.5 gcc 2.6.3 cc
MIPS R3000	Ultrix 4.3	lcc 3.5 gcc 2.5.7
Pentium	Windows 95 Windows NT 3.51	Microsoft Visual C/C++ 4.0

其中几个实现是针对特定机器的。这些实现假设机器使用的是二进制补码表示的整数和 IEEE 浮点算术，并且无符号的长整数可以用来保存对象指针。

本书中所有的源代码在 ftp.cs.princeton.edu 的目录 pub/packages/cii 下，匿名登录就可以下载。使用 ftp 客户端软件连接到 ftp.cs.princeton.edu，转到 pub/packages/cii 目录，下载 README 文件，文件中说明了目录的内容以及如何下载。

大多数最新的实现通常都是以 ciixy.tar.gz 或 ciixy.zip 的文件名存储的，其中 xy 是版本号，例如 10 是指版本 1.0。ciixy.tar.gz 是用 gzip 压缩的 UNIX tar 文件，而 ciixy.zip 是与 PKZIP 2.04g 版兼容的 ZIP 文件。ciixy.zip 中的文件都是 DOS/Windows 下的文本文件，每行均以回车和换行符结束。ciixy.zip 同时也可以在美国在线、CompuServe 以及其他在线服务器上下载。

登录 http://www.cs.princeton.edu/software/cii/ 同样也可以得到相应的信息。该页面还解释了如何报告勘误。

致谢

自 20 世纪 70 年代末以来，在我的科研项目以及在亚利桑那大学和普林斯顿大学的讲课中，我就已经使用过本书中的一些接口。选这些课程的学生最早试用了我设计的这些接口。这些年来他们的反馈凝结在本书代码与说明之中。我要特

别感谢普林斯顿大学选修 COS 217 和 COS 596 课程的学生，正是他们在不知不觉中参与了本书中大多数接口的初步设计。

利用接口开发是 DEC 公司[①]的系统研究中心（System Research Center，SRC）的主要工作方式，1992 年和 1993 年暑假我在 SRC 从事 Modula-3 项目开发，亲身的工作经历消除了我对这种方法有效性的怀疑。我非常感谢 SRC 对我工作的支持，以及 Bill Kalsow、Eric Muller 和 Greg Nelson 与我进行的多次富有启迪的讨论。

我还要感谢 IDA 在普林斯顿的通信研究中心（Center for Communications Research，CCR）和 La Jolla，感谢他们在 1994 年暑假和 1995~1996 整个休假年对我的支持。还要感谢 CCR 为我提供了一个理想的地方，让我从容规划并完成了本书。

与同事和学生的技术交流也在许多方面完善了本书。一些即使看上去不相关的讨论也促使我对代码及其说明做了改进。感谢 Andrew Appel、Greg Astfalk、Jack Davidson、John Ellis、Mary Fernandez、Chris Fraser、Alex Gounares、Kai Li、Jacob Navia、Maylee Noah、Rob Pike、Bill Plauger、John Reppy、Anne Rogers 和 Richard Stevens。感谢 Rex Jaeschke、Brian Kernighan、Taj Khattra、Richard O'Keefe、Norman Ramsey 和 David Spuler，他们仔细阅读了本书的代码和内容，为本书的成功做出了不可磨灭的贡献。

① DEC 公司已被 Compaq 收购。——编者注

CONTENTS

1

INTRODUCTION

A big program is made up of many small modules. These modules provide the functions, procedures, and data structures used in the program. Ideally, most of these modules are ready-made and come from libraries; only those that are specific to the application at hand need to be written from scratch. Assuming that library code has been tested thoroughly, only the application-specific code will contain bugs, and debugging can be confined to just that code.

Unfortunately, this theoretical ideal rarely occurs in practice. Most programs are written from scratch, and they use libraries only for the lowest level facilities, such as I/O and memory management. Programmers often write application-specific code for even these kinds of low-level components; it's common, for example, to find applications in which the C library functions `malloc` and `free` have been replaced by custom memory-management functions.

There are undoubtedly many reasons for this situation; one of them is that widely available libraries of robust, well designed modules are rare. Some of the libraries that are available are mediocre and lack standards. The C library has been standardized since 1989, and is only now appearing on most platforms.

Another reason is size: Some libraries are so big that mastering them is a major undertaking. If this effort even appears to be close to the effort required to write the application, programmers may simply reimplement the parts of the library they need. User-interface libraries, which have proliferated recently, often exhibit this problem.

Library design and implementation are difficult. Designers must tread carefully between generality, simplicity, and efficiency. If the routines and data structures in a library are too general, they may be too hard to

use or inefficient for their intended purposes. If they're too simple, they run the risk of not satisfying the demands of applications that might use them. If they're too confusing, programmers won't use them. The C library itself provides a few examples; its `realloc` function, for instance, is a marvel of confusion.

Library implementors face similar hurdles. Even if the design is done well, a poor implementation will scare off users. If an implementation is too slow or too big — or just perceived to be so — programmers will design their own replacements. Worst of all, if an implementation has bugs, it shatters the ideal outlined above and renders the library useless.

This book describes the design and implementation of a library that is suitable for a wide range of applications written in the C programming language. The library exports a set of modules that provide functions and data structures for "programming-in-the-small." These modules are suitable for use as "piece parts" in applications or application components that are a few thousand lines long.

Most of the facilities described in the subsequent chapters are those covered in undergraduate courses on data structures and algorithms. But here, more attention is paid to how they are packaged and to making them robust. Each module is presented as an *interface* and its *implementation*. This design methodology, explained in Chapter 2, separates module specifications from their implementations, promotes clarity and precision in those specifications, and helps provide robust implementations.

1.1 Literate Programs

This book describes modules not by prescription, but by example. Each chapter describes one or two interfaces and their implementations in full. These descriptions are presented as literate programs. The code for an interface and its implementation is intertwined with prose that explains it. More important, each chapter is the source code for the interfaces and implementations it describes. The code is extracted automatically from the source text for this book; what you see is what you get.

A literate program is composed of English prose and labeled *chunks* of program code. For example,

```
⟨compute x • y⟩≡
  sum = 0;
```