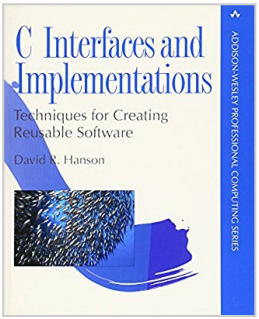

C Interfaces and Implementations: Techniques for Creating Reusable Software

By David R. Hanson

Download now

Read Online

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson

creating reusable software modules; they are the building blocks of large, reliable applications. Unlike some modern object-oriented languages, C provides little linguistic support or motivation for creating reusable application programming interfaces (APIs). While most C programmers use APIs and the libraries that implement them in almost every application they write, relatively few programmers create and disseminate new, widely applicable APIs. C Interfaces and Implementations shows how to create reusable APIs using interface-based design, a language-independent methodology that separates interfaces from their implementations. This methodology is explained by example. The author describes in detail 24 interfaces and their implementations, providing the reader with a thorough understanding of this design approach. Features of C Interfaces and Implementations: * Concise interface descriptions that comprise a reference manual for programmers interested in using the interfaces. * A guided tour of the code that implements each chapters interface tp help those modifying or extending an interface or designing related interfaces. * In-depth focus on algorithm engineering: how to packag

Download C Interfaces and Implementations: Techniques for C ...pdf

Read Online C Interfaces and Implementations: Techniques for ...pdf

C Interfaces and Implementations: Techniques for Creating Reusable Software

By David R. Hanson

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson

creating reusable software modules; they are the building blocks of large, reliable applications. Unlike some modern object-oriented languages, C provides little linguistic support or motivation for creating reusable application programming interfaces (APIs). While most C programmers use APIs and the libraries that implement them in almost every application they write, relatively few programmers create and disseminate new, widely applicable APIs. C Interfaces and Implementations shows how to create reusable APIs using interface-based design, a language-independent methodology that separates interfaces from their implementations. This methodology is explained by example. The author describes in detail 24 interfaces and their implementations, providing the reader with a thorough understanding of this design approach. Features of C Interfaces and Implementations: * Concise interface descriptions that comprise a reference manual for programmers interested in using the interfaces. * A guided tour of the code that implements each chapters interface tp help those modifying or extending an interface or designing related interfaces. * In-depth focus on algorithm engineering: how to packag

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson Bibliography

- Rank: #104964 in Books
- Brand: David R Hanson
- Published on: 1996-08-30
- Original language: English
- Number of items: 1
- Dimensions: 9.10" h x 1.30" w x 7.30" l, 2.03 pounds
- Binding: Paperback
- 544 pages

Download C Interfaces and Implementations: Techniques for C ...pdf

Read Online C Interfaces and Implementations: Techniques for ...pdf

Download and Read Free Online C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson

Editorial Review

From the Back Cover

Every programmer and software project manager must master the art of creating reusable software modules; they are the building blocks of large, reliable applications. Unlike some modern object-oriented languages, C provides little linguistic support or motivation for creating reusable application programming interfaces (APIs). While most C programmers use APIs and the libraries that implement them in almost every application they write, relatively few programmers create and disseminate new, widely applicable APIs. **C Interfaces and Implementations** shows how to create reusable APIs using interface-based design, a language-independent methodology that separates interfaces from their implementations. This methodology is explained by example. The author describes in detail 24 interfaces and their implementations, providing the reader with a thorough understanding of this design approach.

Features of **C Interfaces and Implementations**:

- Concise interface descriptions that comprise a reference manual for programmers interested in using the interfaces.
- A guided tour of the code that implements each chapter's interface tp help those modifying or extending an interface or designing related interfaces.
- In-depth focus on "algorithm engineering:" how to package data structures and related algorithms into reusable modules.
- Source code for 24 APIs and 8 sample applications is examined, with each presented as a "literate program" in which a thorough explanation is interleaved with the source code.
- Rarely documented C programming tricks-of-the-trade.
- Convenient access to all source code in the book via the World Wide Web at http://www.cs.princeton.edu/software/cii/

0201498413B04062001

About the Author

David R. Hanson is a Professor of Computer Science at Princeton University with more than 20 years of research experience in programming languages. He has conducted research in conjunction with Bell Laboratories and is the co-author of `lcc`, a production quality, research compiler for the C language that is popular with the Unix community. lcc is presented and analyzed in the book **A Retargetable C Compiler: Design and Implementation** , by Christopher Fraser and David Hanson (c) 1995, Addison-Wesley.

0201498413AB04062001

Excerpt. © Reprinted by permission. All rights reserved.

Programmers are inundated with information about application programming interfaces, or APIs. Yet, while most programmers use APIs and the libraries that implement them in almost every application they write, relatively few create and disemminate new, widely applicable, APIs. Indeed, programmers seem to prefer to "roll their own" instead of searching for a library that might meet their needs, perhaps because it is easier to write application-specific code than to craft well-designed APIs.

I'm as guilty as the next programmer: lcc, a compiler for ANSI/ISO C written by Chris Fraser and myself, was built from the ground up. (lcc is described in **A Retargetable C Compiler: Design and Implementation**, Addison-Wesley, 1995.) A compiler exemplifies the kind of application for which it is possible to use standard interfaces and to create interfaces that are useful elsewhere. Examples include interfaces for memory management, string and symbol tables, and list manipulation. But lcc uses only a few routines from the standard C library, and almost none of its code can be used directly in other applications.

This book advocates a design methodology based on interfaces and their implementations, and it illustrates this methodology by describing 24 interfaces and their implementations in detail. These interfaces span a large part of the computing spectrum and include data structures, arithmetic, string processing, and concurrent programming. The implementations aren't toys - they're designed for use in production code. As described below, the source code is freely available.

There's little support in the C programming language for the interface-based design methodology. Object-oriented languages, like C++ and Modula-3, have language features that encourage the separation of an interface from its implementation. Interface-based design is independent of any particular language, but it does require more programmer willpower and vigilance in languages like C, because it's too easy to pollute an interface with implicit knowledge of its implementation and vice versa.

Once mastered, however, interface-based design can speed development time by building upon a foundation of general-purpose interfaces that can serve many applications. The foundation class libraries in some C++ environments are examples of this effect. Increased reuse of existing software - libraries of interface implementations - reduces initial development costs. It also reduces maintenance costs, because more of an application rests on well-tested implementations of general-purpose interfaces.

The 24 interfaces come from several sources, and all have been revised for this book. Some of the interfaces for data structures - abstract data types - originated in lcc code, and in implementations of the Icon programming language done in the late 1970s and early 1980s (see R. E. Griswold and M. T. Griswold, **The Icon Programming Language**, Prentice Hall, 1990). Others come from the published work of other programmers; the "Further Reading" sections at the end of each chapter give the details.

Some of the interfaces are for data structures, but this is not a data structures book, per se. The emphasis is more on algorithm engineering - packaging data structures for general use in applications - than on data-structure algorithms. Good interface design does rely on appropriate data structures and efficient algorithms, however, so this book complements traditional data structure and algorithms texts like Robert Sedgewick's **Algorithms in C** (Addison-Wesley, 1990).

Most chapters describe one interface and its implementation; a few describe related interfaces. The "Interface" section in each chapter gives a concise, detailed description of the interface alone. For programmers interested only in the interfaces, these sections form a reference manual. A few chapters include "Example" sections, which illustrate the use of one or more interfaces in simple applications.

The "Implementation" section in each chapter is a detailed tour of the code that implements the chapter's interface. In a few cases, more than one implementation for the same interface is described, which illustrates an advantage of interface-based design. These sections are most useful for those modifying or extending an

interface or designing related interfaces. Many of the exercises explore design and implementation alternatives. It should not be necessary to read an "Implementation" section in order to understand how to use an interface.

The interfaces, examples, and implementations are presented as literate programs; that is, the source code is interleaved with its explanation in an order that best suits understanding the code. The code is extracted automatically from the text files for this book and assembled into the order dictated by the C programming language. Other book-length examples of literate programming in C include **A Retargetable C Compiler** and **The Stanford GraphBase: A Platform for Combinatorial Computing** by D. E. Knuth (Addison-Wesley, 1993).

Organization
The material in this book falls into the following broad categories:

Most readers will benefit from reading all of Chapters 1 through 4, because these chapters form the framework for the rest of the book. The remaining chapters can be read in any order, although some of the later chapters refer to their predecessors.

Chapter 1 covers literate programming and issues of programming style and efficiency. Chapter 2 motivates and describes the interface-based design methodology, defines the relevant terminology, and tours two simple interfaces and their implementations. Chapter 3 describes the prototypical Atom interface, which is the simplest production-quality interface in this book. Chapter 4 introduces exceptions and assertions, which are used in every interface. Chapters 5 and 6 describe the memory management interfaces used by almost all the implementations. The rest of the chapters each describe an interface and its implementation.

Instructional Use
I assume that readers understand C at the level covered in undergraduate introductory programming courses, and have a working understanding of fundamental data structures at the level presented in texts like Algorithms in C. At Princeton, the material in this book is used in systems programming courses from the sophomore to first-year graduate levels. Many of the interfaces use advanced C programming techniques, such as opaque pointers and pointers to pointers, and thus serve as nontrivial examples of those techniques, which are useful in systems programming and data structure courses.

This book can be used for courses in several ways, the simplest being in project-oriented courses. In a compiler course, for example, students often build a compiler for a toy language. Substantial projects are common in graphics courses as well. Many of the interfaces can simplify the projects in these kinds of courses by eliminating some of the grunt programming needed to get such projects off the ground. This usage helps students realize the enormous savings that reuse can bring to a project, and it often induces them to try interface-based design for their own parts of the project. This latter effect is particularly valuable in team projects, because that's a way of life in the "real world."

Interfaces and implementations are the focus of Princeton's sophomore-level systems programming course. Assignments require students to be interface clients, implementors, and designers. In one assignment, for example, I distribute Section 8.1's Table interface, the object code for its implementation, and the specifications for Section 8.2's word frequency program, wf. The students must implement wf using only my object code for Table. In the next assignment, they get the object code for wf, and they must implement Table. Sometimes, I reverse these assignments, but both orders are eye-openers for most students. They are unaccustomed to having only object code for major parts of their program, and these assignments are usually their first exposure to the semiformal notation used in interfaces and program specification.

Initial assignments also introduce checked runtime errors and assertions as integral parts of interface

specifications. Again, it takes a few assignments before students begin to appreciate the value of these concepts. I forbid "unannounced" crashes; that is, crashes that are not announced by an assertion failure diagnostic. Programs that crash get a grade of zero. This penalty may seem unduly harsh, but it gets the students' attention. They also gain an appreciation of the advantages of safe languages, like ML and Modula-3, in which unannounced crashes are impossible. (This grading policy is less harsh than it sounds, because in multipart assignments, only the offending part is penalized, and different assignments have different weights. I've given many zeros, but none has ever caused a course grade to shift by a whole point.)

Once students have a few interfaces under their belts, later assignments ask them to design new interfaces and to live with their design choices. For example, one of Andrew Appel's favorite assignments is a primality testing program. Students work in groups to design the interfaces for the arbitrary-precision arithmetic that is needed for this assignment. The results are similar to the interfaces described in Chapters 17 through 19. Different groups design interfaces, and a postassignment comparison of these interfaces, in which the groups critique one anothers' work, is always quite revealing. Kai Li accomplishes similar goals with a semester-long project that builds an X-based editor using the Tcl/Tk system (J. K. Ousterhout, **Tcl and the Tk Toolkit**, Addison-Wesley, 1994).

0201498413P04062001

Users Review

From reader reviews:

Mary Oropeza:

Why don't make it to be your habit? Right now, try to prepare your time to do the important behave, like looking for your favorite reserve and reading a e-book. Beside you can solve your long lasting problem; you can add your knowledge by the e-book entitled C Interfaces and Implementations: Techniques for Creating Reusable Software. Try to the actual book C Interfaces and Implementations: Techniques for Creating Reusable Software as your buddy. It means that it can to be your friend when you sense alone and beside associated with course make you smarter than ever before. Yeah, it is very fortuned in your case. The book makes you far more confidence because you can know every little thing by the book. So , we need to make new experience and knowledge with this book.

Phillip Barker:

Hey guys, do you would like to finds a new book you just read? May be the book with the name C Interfaces and Implementations: Techniques for Creating Reusable Software suitable to you? Often the book was written by well known writer in this era. The book untitled C Interfaces and Implementations: Techniques for Creating Reusable Softwareis a single of several books which everyone read now. This particular book was inspired many men and women in the world. When you read this guide you will enter the new dimensions that you ever know just before. The author explained their strategy in the simple way, thus all of people can easily to know the core of this book. This book will give you a large amount of information about this world now. To help you see the represented of the world with this book.

Kelly Blow:

Reading can called mind hangout, why? Because when you are reading a book specially book entitled C Interfaces and Implementations: Techniques for Creating Reusable Software your head will drift away trough every dimension, wandering in every aspect that maybe unfamiliar for but surely might be your mind friends. Imaging each word written in a book then become one contact form conclusion and explanation which maybe you never get before. The C Interfaces and Implementations: Techniques for Creating Reusable Software giving you yet another experience more than blown away the mind but also giving you useful info for your better life in this era. So now let us explain to you the relaxing pattern is your body and mind will be pleased when you are finished reading it, like winning a casino game. Do you want to try this extraordinary shelling out spare time activity?

Allen Green:

That reserve can make you to feel relax. This kind of book C Interfaces and Implementations: Techniques for Creating Reusable Software was colorful and of course has pictures around. As we know that book C Interfaces and Implementations: Techniques for Creating Reusable Software has many kinds or category. Start from kids until teenagers. For example Naruto or Detective Conan you can read and think you are the character on there. Therefore not at all of book tend to be make you bored, any it offers up you feel happy, fun and rest. Try to choose the best book for you and try to like reading this.

Download and Read Online C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson #317PYFC4LKO

Read C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson for online ebook

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson Free PDF d0wnl0ad, audio books, books to read, good books to read, cheap books, good books, online books, books online, book reviews epub, read books online, books to read online, online library, greatbooks to read, PDF best books to read, top books to read C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson books to read online.

Online C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson ebook PDF download

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson Doc

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson Mobipocket

C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson EPub

317PYFC4LKO: C Interfaces and Implementations: Techniques for Creating Reusable Software By David R. Hanson