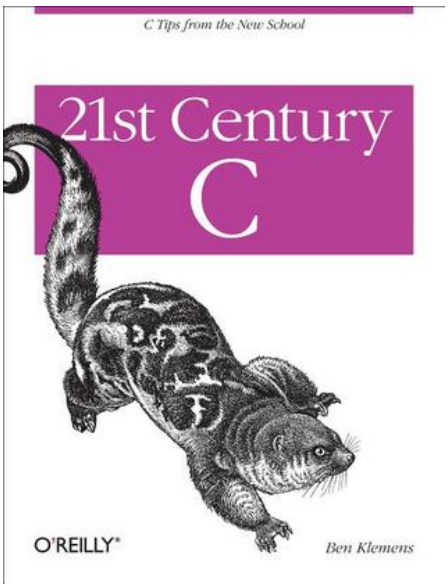

21st Century C: C Tips from the New School

Ben Klemens

21st Century C: C Tips from the New School

Ben Klemens

21st Century C: C Tips from the New School Ben Klemens
Throw out your old ideas of C, and relearn a programming language that's substantially outgrown its origins. With *21st Century C*, you'll discover up-to-date techniques that are absent from every other C text available. C isn't just the foundation of modern programming languages, it is a modern language, ideal for writing efficient, state-of-the-art applications. Learn to dump old habits that made sense on mainframes, and pick up the tools you need to use this evolved and aggressively simple language. No matter what programming language you currently champion, you'll agree that C *rocks*.

Set up a C programming environment with shell facilities, makefiles, text editors, debuggers, and memory checkers
Use Autotools, C's de facto cross-platform package manager
Learn which older C concepts should be downplayed or deprecated
Explore problematic C concepts that are too useful to throw out
Solve C's string-building problems with C-standard and POSIX-standard functions
Use modern syntactic features for functions that take structured inputs
Build high-level object-based libraries and programs
Apply existing C libraries for doing advanced math, talking to Internet servers, and running databases

21st Century C: C Tips from the New School Details

Date	:	Published November 5th 2012 by O'Reilly Media (first published January 1st 2012)
ISBN	:	9781449327149
Author	:	Ben Klemens
Format	:	Paperback 296 pages
Genre	:	Computer Science, Programming, Reference, Science, Nonfiction, Technology, Computers

From Reader Review 21st Century C: C Tips from the New School for online ebook

Russ says

Lots of very good information that it took me years working professionally to figure out. Source control, libtool, automake, doxygen, this book covers them all.

Vasil Kolev says

This was bad. There was nothing new for me in there, except probably asprintf() (and with a title like this I expected a lot more), and all the stuff explained was brief, non-systematic and mostly an explanation of the author's mode of work. Everyone would be better by reading the unix network programming books by Stevens, and a few manuals for whatever he/she sees used in any medium-sized project.

Dee says

I've read it all and I keep reading parts of this great little book. Probably the best,most helpful book I've read about programming in the C language right after "The C Programming Language," 2nd edition, by Kernighan and Ritchie.

// just a snippet of an imaginary program I didn't write. I think it's correct - I used stuff from the
// book to write it. I didn't know about the brackets, f'rinstance.

/*
the double slash is a new way of commenting in C, to annotate the program
and the other, established way is to bracket the non-C text between a slash, then asterisk, at the start and an asterisk, then a slash at the end. You hit return in between. The '//' method is for one line only
*/
char **review ={"this variable is in scope but", "after the open bracket, it goes out of scope",
"until the close bracket"}
{
char **review[]={"I learned some really good stuff", "understanding scope is very useful and I learned it in this book", "also the forms of variable and the usage of the terms denoting them, which changes by the context of the program they're declared in is extremely USEFUL and IMPORTANT to know"};

/* also, important and use full information about multithreaded programming, oh, the list goes on and on. I'm using what I've learned a LOT. Learning what tail end recursion is and how it works was like being born again, a whole new beginning. And it sped up my program a bunch, too*/

Chris Maguire says

This book is awesome. The author is experienced, clear and funny. Klemens' tactic is to give a crash course in the various tools needed to produce usable, portable, up-to-date C code without getting into boring minutiae that can be googled when needed. To me this book means the difference between being able to compile an executable and being able to write software. I'm actually excited to write some C after this, but also a little scared.

Word of warning: at least a little knowledge of C is required. I did a little C in college back in '98 and I was able to pick most of this up without too much trouble.

Colin P. says

Weird rock music references and some of it just feels wrong but some valuable advice within.

James Anderson says

In aggregate, this is kind of a mixed bag, and I wouldn't recommend it to anyone who hasn't spent at least some time learning C the old fashioned way, but for anyone who has, there's a lot of really interesting tips that don't get covered in traditional sources, such as things that made sense to do in the early 80s that are mostly irrelevant on modern machines. The easygoing style is very much appreciated, and I think really sets it apart from most technical works of this nature in making it easy to read. The underlying theme is that "C is punk rock", which is laid out in the introduction, and continued with relevant punk rock song lyrics in the chapter openings and advice that runs counter to the mainstream.

As for downsides, one is that some tips are platform-specific or not considered "best practices", though to be fair they are always marked as such with a caveat. Second, the author has a serious axe to grind with "bloggers", who are often mentioned as being extremely pedantic and frowning at the sort of advice presented in the book. I'm somewhat aware of the culture the author is pushing back against, but the barbs seem a little too pointed and directed at specific (though unnamed) authors for an otherwise professional work. Third, the first couple of chapters are really painful (compiling and the endless series of flags necessary to do so, testing, debugging, packaging...), and the fun stuff really starts in Part II. I understand the necessity of these things, but I can't bring myself to finish any of them quite yet, and I ended up skipping ahead to the parts about the language itself.

This came out sounding more negative than I'd intended, but ultimately, warts aside, this book is very helpful and I'd recommend it to anyone hoping to better understand modern C programming.

Sefa says

Nice idea, poor implementation.

This book contains two parts: tools supplementary to C programming and a collection of C tips on pointers, macros, functions returning multiple items, functions that take a variable number of parameters.

The first part -tools useful for C programming such as makefile, debugger (gdb), memory profiling (Valgrind), version control (git)- is an ambitious move, that is trying to cover all these tools. These tools are mostly from past century, they are powerful yet sometimes not easy to use. Each of these tools deserves exclusive books and there are books on each of them. Unfortunately, this book covers them in a very unorganized way. It is hard to get a sense of what each tool is for and what it is capable of doing.

The second part provides some useful tips on C, how to do things that are so easy in some other high-level languages that make you jealous. Some of them are better string handling, mapping a function over a set of data items, flexible function inputs, etc.

I had much higher expectations from this book. Now we need to wait another 10-15 years to have another C book :)

Yung-jin says

21st Century C is one of those rare technical books that is truly what I would consider to be an intermediate level book. It is a pity that more books aren't written at this level, as this is where I feel like I currently am in my career.

This is not an "Into to Programming" book that happens to use C. Instead it is a book that a moderately experienced non-C programmer can use to learn some C. Not everything is spelled out step-by-step. You'll need to look up the documentation to some of the libraries that are used, and cruse around StackOverflow to figure out why things aren't compiling. If you've been a programmer for a few years (in say a language like Python) you'll be right at home with this workflow.

In the end, you get out of this book what you put into it. Type out the examples, and get them to work on your machine. You will inevitably typo a few bugs, and nothing helps you learn more about a programming language than debugging.

This may also be a great "next step" book for beginner C programmers once they're done with an "Intro to C" book. I jumped into it directly after a few years of Python, C#, and C++11 experience, and it was a great ride.

---

gekko100 says

If you want to learn to program in C, the time-honoured recommendation is Kernighan & Richie's venerable and succinct text "The C Programming Language". Like the language itself, that book is now getting a bit long in the tooth, and fails to cover the more recent revisions to the language (C99 and C11), not to mention the evolution in operating systems and development environments.

"21st Century C" steps in to fill this decades-long gap and provides an up-to-date perspective on programming in the C language that would make sense to someone stepping out of a time machine from the earliest days of ANSI C89. Having started working through the first few chapters of K&R, I have found this book to be an ideal accompaniment.

The first five chapters provide an overview of the ecosystem of tools that exist to make life bearable for the modern programmer: the command line, package managers, debuggers (gdb, valgrind), version control and

so on. This is the kind of stuff that is absorbed by osmosis and quickly becomes second nature to any self-respecting developer; for a dilettante like me, on the other hand, it's very useful to have it all described in one place.

The remainder of the book (Chapters 6 to 13) discusses various aspects of the language and programming style. I have yet to read to this part of the book but at a first glance looks like it will be well worth revisiting, probably when I finish reading Kernighan & Richie.

Cristian says

I liked the book's premise, and it has some interesting snippets and gems, but unfortunately it falls short in a couple of areas.

Having information and examples on the surrounding tooling is certainly practical, but the git chapter felt really unnecessary. Git is so ubiquitous nowadays, that I hardly see the point. Including the other tools, however, makes more sense (valgrind, doxygen, make, autotools, etc.). Instead of Git, the author could have played with some of the available static analysis tools, or linters.

A lot of the code samples aren't properly formatted, which is a shame because it gives the impression of carelessness. The book is expected to be an advanced or at least intermediate C book, and as such it doesn't waste much time with the basics. That is all the more reason for the code to have a consistent style throughout. People read it to learn from it, and being inconsistent just sets a bad example. All the author needed was to run a formatter (e.g. clang-tidy) on the code samples.

It also uses the anti-idiom of doing:

```
int *i = malloc(sizeof(int));
```

instead of:

```
int *i = malloc(sizeof(*i));
```

This is an old idiom, recommended in books and by the SEI CERT C Coding Standard [1].

All in all, reading it was worth it, but the book could be improved for a third edition by fixing the aforementioned issues, and expanding on some of the 21st Century C wisdom, such as that available in the SEI CERT C Coding Standard, or Jens Gustedt's Modern C.

Reference:
[1] https://www.securecoding.cert.org/con...

Marshall says

The idea of this book is that the C programming language has been quietly evolving, while C++ and Java took all the glory. But those languages are a mess, C++ in its syntax and gotchas and Java in its resource abuse. In its recent standards and libraries, C has many of the same features, rendering the newer languages mere syntactic sugar. C doesn't candy coat. Or, as this book puts it, it's the "punk rock of programming

languages." Fast, messy, and dangerous.

I thought it was an interesting idea, worth hearing him out. It did give some good tips, and helped re-awaken my understanding of C, which has been atrophying in my mind for decades. Most notably, it helped me understand C idioms that have evolved over time, which I know is just as important is knowing the syntax, if you want to have any hope for quickly understanding code other people write. However, this book also reminded me why I hate this language. It fights you every step of the way, turning even simple tasks into an ordeal. It encourages a heavy use of pointers, which turns the code into a sea of asterisks. The book is pretty straightforward to read, but it does have a bad habit of dumping pages of code, with little explanation. I guess the author figures a book on C should reflect the attitude of the language itself.

I'm resolved to the reality that C is the basis for most software and programming, and that isn't changing anytime soon, if ever. It's simply the only realistic option if you want to write low-level code. No matter how advanced the hardware gets, there will always be a need for speed and control, so there will always be a need for C. Might as well get more comfortable with the language, which means updating our understanding of it.

UPDATE: I first read this a couple years ago, and I just finished reading it again. Since the first reading, I've been through a huge undertaking of discovery and learning about programming languages. There were over 20 language I've explored. It's been a lot of fun, and those languages have a lot to offer, but you want to hear something funny? I just kept going back to C. Each time I did, I had a deeper understanding of it. C was the first language I'd written any serious code in, so it felt like coming home. I craved the control it gave me. It feels like talking to the computer directly, rather than having a translator. 15 years after rejecting this language, I'm coming back to it. In these modern times of Ruby, Python, Scala, Haskell, Rust, C++, D, Go, blah blah blah, this book showed me how to modernize C, and get some of the features I crave from it. C is still not pretty, and it can be a bit tedious, but it's FAST, it's simple, and I'm always in the driver's seat. This book deserves the credit for this reawakening, so I'm upping my rating to 5-stars.

Louis says

All engineering students take a one semester course fairly early in their education. My experience as a professor is that most of them promptly forget it and never make use of it again. I get alternately amused and discouraged at the lengths some of the students I work with go to avoid programming again. Because that one semester course teaches a programming language, but does not teach someone how to program. This book is meant to take someone through that step. It is sometimes hard to follow (my heavy programming in C was years ago) but it gets someone to where they can be productive.

When someone is referred to me an claims to have some programming background, I have learned to ask a simple question: have you programmed with libraries? I usually get a blank stare, and I know that this person may have taken a class, but cannot do anything useful. What this is how to make programming in C useful. So it builds in working with the various utilities that make C programming useful (I learned pkg-config and profiling with valgrind with this book. And before I used to be able to edit Makefiles but I would not try to write one from scratch) And, of course, working with C using some fairly useful libraries (if I am programming I am general pulling data from databases and using numerical methods, so I am rather pleased at the use of SQLite and GSL as example libraries).

The various tools are very useful. Programming without learning to use the various utilities such as debuggers, library managers, packaging, etc. is an exercise in frustration (and is how the standard introduction to programming course is set up).

Am I a convert? I still regard Python and R as my preferred environments, and I'm not swayed by his arguments that C is a "punk" language. I still find that all the little cruft such as pointers and their management something I'm glad to avoid. But I also regard one of the best features of Python and R the fact that I can dive into C when I need something fast and efficient. So being able to do this more effectively is good to improve my toolkit.

The book is not really for pure beginners. If you did not already have some background in C, you need to get that somewhere else. I got in trouble with some of the examples, so it helped that I have used many of these tools before (even if not all that effectively). It also is opinionated, presenting one way to do things. (of course, that is very useful to someone starting out, knowing one good way is better than having a dozen options in front of you when you don't know what is what). But it can be very good for someone who needs to go from "I have learned C" to "I know how to effectively use C."

Note: I received a free electronic copy of this book from the O'Reilly Bloggers program.

Dan Watts says

Half of this book is an enjoyable up-to-date primer on the C programming language. The other half covers 3rd party libraries and tools which are needed to plug gaps in C's built-in capabilities, undermining the author's claim that C is still viable as a development platform. I haven't coded much in C for decades, having moved on to full-featured platforms built around other languages . As a relatively easy-to-read refresher on C, the book works well. As an argument in favor of using C for general purpose programming today, it fell short for me.

JManInPhoenix says

I can see why the reviews for this book are all over the place. This book is more for an experienced C programmer that is actively writing C code for a living. I am neither a working programmer nor a noob but still found quite a few good nuggets throughout the book. I especially liked the coverage of GDB, pointers, what C items should be taught less and what C items should be taught more. For those four things, I give this book five stars as I got useful information from each topic.

Not being a working programmer (just a hobbyist) that codes for a living, most of the tools discussed were not something I was interested in. Other than makefile & GDB, I don't personally use any of the tools discussed.

Tudor ?tef?nescu says

o trecere în revist? a ceea ce-?i trebuie pentru a programa în C. Nu intr? în am?nunte dar e bun? ca s?-?i dea câteva idei de ce ar trebui s? cau?i mai departe, iar pe alocuri chiar o po?i folosi ca pe un "quick guide" (în special partea de source control - git). În plus se concentreaz? pe "noul C": C99/C11 care are câteva lucruri pe care eu (programator C/C++ de 11 ani) nu le ?tiam (cum ar fi keyword-ul "restrict").
