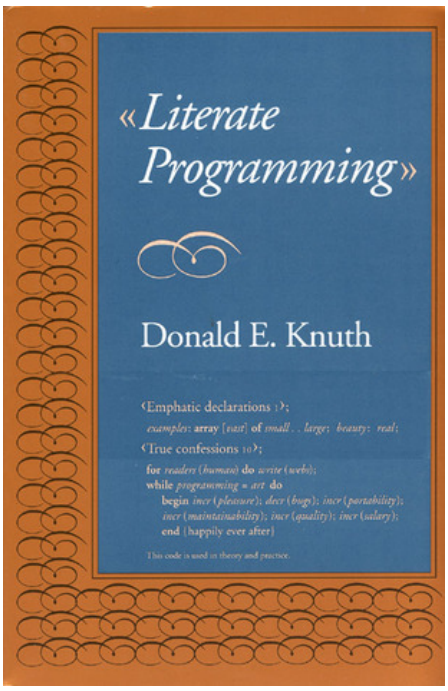

Literate Programming

Donald Ervin Knuth

Literate Programming

Donald Ervin Knuth

Literate Programming Donald Ervin Knuth
This anthology of essays from Donald Knuth, "the father of computer science," and the inventor of literate programming includes early essays on related topics such as structured programming, as well as *The Computer Journal* article that launched literate programming itself. Many examples are given, including excerpts from the programs for TeX and METAFONT. The final essay is an example of CWEB, a system for literate programming in C and related languages.

This volume is first in a series of Knuth's collected works.

Literate Programming Details

Date	:	Published June 1st 1992 by Center for the Study of Language and Inf (first published March 1st 1992)
ISBN	:	9780937073803
Author	:	Donald Ervin Knuth
Format	:	Paperback 384 pages
Genre	:	Computer Science, Programming, Science, Software, Computers, Nonfiction, Technical

From Reader Review Literate Programming for online ebook

Peter Aronson says

Three-and-a-half stars? I'm not entirely sure what to think about this book. On one hand, it did make me think, and some of the articles, such as "Computer Programming as an Art" and "The Errors of TEX" were particularly good. But the central thesis of the book, literate programming, just did not work for me, at least not in the examples provided. Frankly I did not find Knuth's literate programs either fun to read, or more easily understood than well designed code with well chosen variable and function names would have been (and far too many of Knuth's variable names are obscure). The reordering of code didn't really do anything for me, nor did I find the use of special symbols to enhance a program's readability. But it works for some people -- the approach still has its adherents, and I certainly feel more in sympathy with them than with the "comments as an antipattern" crowd.

Dave says

Classic essays on improving the state of programming literature.

Enrico says

A paradigm shift for how humans should describe what they intend the computers do.

Saharvetes says

This is a collection of material by Knuth about the philosophy and practice of **Literate Programming**, his idea of programs as literature. All programmers today (claim to) understand the importance of readable code, and slogans like "Programs must be written for people to read, and only incidentally for machines to execute" are occasionally heard from several quarters; however, with most programmers, such noble intentions seem to stop at (the intention of) writing a lot of comments to document their code. It is only the *literate programming* system that really takes the idea seriously and to its logical conclusion, by (1) freeing the programmer to write code in whatever order is appropriate for exposition, not the order that the compiler wants, and, more generally, (2) being a system for writing a document that *contains* the program, not for documenting a program after it has been written. The first feature means that literate programming can be useful *even if you don't write a single line of comments.*

This book, like the books in Knuth's "Selected Papers in…" series, is a collection of disparate papers and articles written over many decades, each of which has some some connection (in the author's mind at least) to literate programming.

The first three chapters are *Computer Programming as an Art* (1974, Knuth's Turing Award lecture), *Structured Programming with go to Statements* (1974, a very long paper discussing several topics that were being debated at the time as part of the Structured Programming revolution), and a short *A Structured Program to Generate All Topological Sorting Arrangements* (1974, written with Jayme L. Szwarcfiter). These were all written before the idea of "literate programming" had been conceived, but they give insights

into Knuth's philosophy of programming.

Chapters 4 to 7 are the real meat of the book, or at least what a reader may expect from the title: Chapter 4 is *Literate Programming* (1984), the original article that introduced Literate Programming; Chapter 5 is *Programming Pearls: Sampling* (1986) by Jon Bentley, in which Bentley introduced Literate Programming to the readers of the *Communications of the ACM* through his column; and Chapter 6 is *Programming Pearls, Continued: Common Words* (1986), the follow-up column in which Knuth wrote a literate program as asked, accompanied by a *review* by Doug McIlroy. Literature deserves literary criticism, and McIlroy sets a wonderful example... I wish program reviews had become a regular feature. Chapter 7 is *How to Read a WEB* (1986), another brief introduction to Literate Programming meant for readers who mainly wish to *read* literate programs written in WEB.

Chapter 8 consists of *Excerpts from the Programs for TEX and METAFONT* (1986). Chapter 9 has excerpts from *Mathematical Writing* (1987), another wonderful book that came out of a course on mathematical writing at Stanford taught by Knuth. It consists of informal class discussions on programming style and examining real literate programs written by other students, written by a student, Paul Roberts. Finally, Chapter 10 *The Errors of TEX* (1989) is an article where Knuth analyses all the bugs that were ever fixed in TeX, from the trivial to the serious. He attributes TeX's being bug-free (besides opinions held by some that TeX itself is a bug) to literate programming, and this analysis was possible because he kept a detailed log, which is reproduced as Chapter 11: *The Error Log of TEX* (1978–1991).

Finally, Chapter 12 *An Example of CWEB* (1990) contains one of the first examples of literate programming outside the WEB system of TeX+Pascal, since CWEB is TeX+C. They write a drop-in replacement to Unix's wc program which counts words, lines or characters.

All in all, it's a good book, especially for those who find almost everything written by Knuth deeply enjoyable. Even in sections of the book that are unrelated to literate programming per se, Knuth's example programs contain clever algorithms and data structures that can be instructive. It is not an ideal book for evangelizing literate programming, however; one wishes that someone, preferably Knuth, had written a book that is not so tied to the particular combination of Pascal and TeX (with its documentation designed for paper, with page numbers and indexes), unfamiliar to the majority of programmers today.

Xavier Shay says

Had been sitting on my shelf at work for a while, finally polished it off.

Knuth is a pretty incredible dude. He wrote TeX without even running it and basically nailed it. I didn't really take anything away from the book though. Everything is so totally old school it's hard to relate, and writing programs to be read is SO MUCH EASIER with the languages we have today. I feel pretty spoiled, actually.

Eugene Miya says

Post-Tex and Metafont and pre-Selected Papers, Knuth experimented with a document system he named Web, quite a few years before Tim Berners-Lee wrote a little system at CERN on his NeXT box. Web, like TeX, has a detailed, assembly language-like view of documents which we might view as "quaint" today, but little gems still exist in this book.

By far and away in the most important relevant chapter (to this day) is the guest oyster column of Jon Louis Bentley's brief *Communications of the ACM* column "Programming Pearls" wherein Don writes a "little" table building program in Pascal. It would be enough but the real gem which every programmer needs to read and understand is Doug McIlroy's "rebuttal" which practically demolishes Knuth's faberge egg. I think Don grinned a great smile when friends show him to be "human."

Programmers should own this book.

Max Lybbert says

I've recently read books by several mathematicians/programmers, such as Dijkstra and Stepanov and some research papers from Jet Propulsion Labs. The advantages mathematicians have when it comes to programming are experience with ways to precisely describe algorithms and logic and experience reasoning from first principles.

Knuth is another famous mathematician/programmer. While literate programming avoids formal descriptions and math-like notation, the underlying philosophy *is* a general requirement to describe algorithms and logic in a conversational way. In other words, literate programming promises many -- though not all -- of the benefits of formal methods without some of the hassle.

The essays on structured programming are pure bonuses. Sure, all modern programming languages are designed around structured programming principles. However, it's good to be reminded of what made structured programming such an improvement: the formalization and enforcement of invariants.

Don't skip "The Errors of Tex" chapter, either. It includes significant useful and pragmatic programming advice.

Guy says

Knuth has an interesting idea here, one I'd like to explore further. One I have been exploring further. Frankly, however, it's underdeveloped in this book. He can hardly be faulted for that; the idea is from an early time, before HTML and other modern markup sophistication. What I found most surprising was, considering how dedicated to beautiful documentation this creator of TEX is, his literate programming notation, like his typesetting notation, is simply not beautiful.

I think there is lots of untapped potential in this idea. A modern retelling plus enhancements could relegate this first pass to a footnote of ancestral acknowledgement. If literate programming ever takes off, it won't look quite like this.

Josh Berry says

For the most part, I will assume that folks picking up this book are already predisposed to like the ideas. I will say that it provides a very nice historical context into how software creation has progressed. In particular, the essay on "goto" statements was a lot more fun than would have been expected.

Highlights definitely include the Programming Pearls essay, the final CWEB example, and the retrospective on TeX bugs. The retrospective is fascinating to get an idea of just how Knuth approaches software construction. I have since bought the TeXbook and related material to see any insights that gives. It would be a delight to see how he sketched the high level design of the system.

I should note that Knuth goes out of his way to say that Literate Programming *will not* lead to error free software. Rather, the point is that this style can more easily motivate others (including your future self) to read an implementation, such that they can more easily find the bugs.

This is an argument that is very appealing to me. Especially in what I perceive as the current environment where the popular currents feel that they are trying to remove all "considered harmful" practices such that they can not be used even for valid uses. I look forward to at least trying these techniques on more of my side projects, while keeping the dream of using it on a large one.

Pavel says

I really wanted to like this whole book, it is nicely typeset, looks great, is organized, clear, etc, but I ended up not liking it a whole lot.

The main premise of the book is to do Literate Programming, and that it will prevent bugs and is the right way to write code. This is not true; it is useful in certain instances when applications are small and are very algorithmically intensive. Nowadays a lot of code written is not intensive enough where every snippet/function has to be documented.

The underlying reasons for why to do Literate Programming are still valid: try something to make your craft better. There are many examples in this book of what Knuth did aside from Literate Programming. One such example is keeping a good track of bugs to see which ones happen most often, and then try to figure out how to prevent them.

In summary this book is a good read to see how one smart guy a while ago (in computer terms) tried to improve his code writing ability.

Ruben says

I specially liked "The errors of TeX"
