# An Interview with Donald Knuth

DDJ chats with one of the world's leading computer scientists

By Jack Woehr

For over 25 years, Donald E. Knuth has generally been considered one of the world's leading computer scientists. Although he's authored more than 150 publications, it is Knuth's three-volume *The Art of Computer Programming* which has become a staple on every programmer's bookshelf. In 1974, Knuth was the recipient of computer science's most prestigious prize, the Turing Award. He also received the National Medal of Science in 1979.

In addition to his work developing fundamental algorithms for computer programming, Knuth was a pioneer in computer typesetting with his TEX, METAFONT, and WEB applications. He has written on topics as singular as ancient Babylonian algorithms and has penned a novel. Knuth currently is Professor Emeritus at Stanford University.

Born in Milwaukee, Knuth exhibited an early aptitude for patterns, as evidenced by his creation of crossword puzzles for the school newspaper. This ability culminated in the eighth grade when Knuth won a national contest sponsored by a candy manufacturer. According to Dennis Shasha and Cathy Lazere, authors of *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*, the challenge was to compose as many words as possible using the letters in the phrase "Ziegler's Giant Bar." The judges had about 2500 words on their master list; Knuth came up with approximately 4500 words without using apostrophes.

Still, it was music that Knuth chose to study at Case Institute (later Case Western Reserve) until he was offered a physics scholarship. This lead him to his first encounter with a computer in 1956—an IBM 650.

In this interview, Knuth chats with frequent DDJ contributor Jack Woehr about these and other topics.

**Q: What distinguishes a "computer scientist" from a "computer programmer?"**

The difference between a computer programmer and a computer scientist is a job-title thing. Edsgar Dijkstra wants proudly to be called a "computer programmer," although he hasn't touched a computer now for some years. He wrote his really terrific essay on the *Humble Programmer* discussing this. To me, "computer programmer" is an honorable term, but to some people a computer programmer is somebody who just follows instructions without understanding what he's doing, one who just knows how to get through the idiosyncrasies of some language.

To me, a computer scientist is somebody who has a way of thinking, which resonates with computer programming. The way a computer scientist views knowledge in general is different from the way a mathematician views knowledge, which is different from the way a physicist views knowledge, which is different from the way a chemist, lawyer, or poet views knowledge.

There's about one person in every fifty who has this peculiar way of viewing knowledge. These people discovered each other at the time computers were born. There's a profile of different intellectual capabilities which makes somebody resonate, which makes somebody really in tune with computer programming.

There were computers in the 19th century, the 17th century . . . I imagine there are computer scientists in the pygmy forest. I haven't really carried this out as an experiment, but I imagine that people may not have machines but one in fifty of them, wherever you go, has this profile, this ability. I'm not a sociologist, nor an anthropologist, but reading publications, reading literature, I can sense how much people think like I do, even if they were writing from a different century.

This is the true explanation of why computer science became a university department so fast all around the world. The reason is not that computers are important tools for mankind, or something like that. The reason is that there were these people out there who had this way of thinking that never had a home before. They get together, they can communicate high bandwidth to each other, the same kind of analogies are meaningful to them. All of a sudden they could come together and work much more efficiently, not in someone else's territory that wasn't for them.

There was a time when there were no physics departments, there was "Natural Philosophy," which combined all kinds of different skills. Over the years, these strong areas of focus materialize, become recognized, and then they get a name. "Computer Science" happens to be one of the more recent ones to crystallize in this way.

**Q: When can we buy volume four of *The Art of Computer Programming*?**

I'm going to be putting it out 128 pages at a time, about twice a year over the next eight years. I'm estimating it now at a little more than 2000 pages. There will be volume 4-a, volume 4-b, and volume 4-c. Volume 4 in general is combinatorial algorithms. Volume 4-a is about finding all possibilities: There's a lot to be said about generating them in good ways—problems where finding all reasonable solutions is not a trivial task. 4-b is going to be about graph and network algorithms, and 4-c is about combinatorial optimization. So 4-a is "find all arrangements," 4-b is "find arrangements that have to do with graphs and networks," and 4-c is "find the best arrangement."

Into those 2000 pages, I have to compress about 200,000 pages of literature. I've been working on it a long time.

**Q: Is the hiatus between volumes 3 and 4 writer's block, that you say, "If I study this more . . . "**

No, that's a pretty good hypothesis. But I had started volume 4 and then realized I had to work on typography. There was a revolution in the printing industry. The printing industry became computer science. It changed from metallurgy to bits, to 0s and 1s. There was no way to print my books with the quality they had before.

I was going to take a year and give a computer scientist's answer to how to print books, and that took ten years. The systems are in common use today. So I think I'm going to be able to recoup that. It wasn't that I didn't have anything to say in volume 4, but that I had this other thing to do that was very timely. My students and I worked very hard on the desktop-publishing revolution.

**Q: One goes to accomplish a task on the computer and realizes that to finish it requires another task, and to finish that one requires another. . . .**

Niklaus Wirth always wanted to design airplanes, but he needed a better tool, so he designed many computer languages and microcomputers and so on. I needed to write my books in some way that would be immune to changes in technology, to capture the book in some electronic form that wasn't going to change when the printing establishment changed.

**Q: I've read the laments of the memorizers of Egypt that were recorded by the scribes when introduced in the ancient kingdom. You're one who is not going to curse the darkness, you're getting ready for the time when books aren't printed anymore.**

I have books that are going to exist no matter how the technology changes.

**Q: Has your digression into TEX and MetaFont been profitable, as well as rewarding?**

I put it in the public domain, but I do get royalties on the books. I give one-third of those to the user group. The important thing, once you have enough to eat and a nice house, is what you can do for others, what you can contribute to the enterprise as a whole.

**Q: I wonder if that same philosophy informs your scientific discipline. To the vast majority of the computer literate, you made a large contribution in stating once and for all how one, for instance, divides two binary numbers efficiently. A programmer wonders how to do something and reaches over his or her shoulder for "knuth," it's like reaching for a "thermos."**

I tried to write things up in a way that was jargon-free, so the nonspecialist would understand it. What I succeeded in doing is making it so that the specialist can understand it, but if I hadn't tried to write jargon-free, then I would have written for specialists, and then the specialists wouldn't be able to get it either. So I've been reasonably successful in boiling down a large volume of material, but still my books aren't easy reading except for specialists.

I'm about to publish a book, S*elected Papers in Computer Science*, which is a collection of papers I've written over the years for nonspecialists. It's being published jointly by the Center for the Study of Linguistics at Stanford (CLSI) and the Cambridge University Press. It has 15 chapters, each of which was an expository lecture. I enjoyed reading them again, though I've edited them to take out sexist language! I think this book is something that might be of interest to the scientific community as a whole.

I plan eight books collecting all the papers I've written: There's going to be *Selected Papers in Analysis of Algorithms*, *Selected Papers in Digital Typography*, *Selected Papers in Fun and Games*, *Selected Papers in Programming Languages*, and so on. This is the second volume; the first volume was *Literate Programming*.

**Q: What has the reward you offer for finding bugs in TEX reached? I had heard it was up to $40.96.**

Oh, that! The reward doubled every year until it reached $655.36 and I froze it at $2^{16}$ pennies. It wouldn't take another ten or fifteen years before it began to exceed the GNP, you know! I paid out a couple of those last February.

**Q: That was the problem posed by the inventor of the chessboard in ancient India, who asked for one grain of wheat on the first square, two on the second, four on the third . . .**

It was al-Biruni in 10th-century Baghdad who explained how to calculate $2^{64}$ efficiently by repeated squaring.

**Q: This is one of the computer scientists of other eras about whom you spoke earlier.**

He was definitely a computer scientist. He knew how many grains of wheat there were without doubling 64 times. al-Khwârizmî, who lived about 150 years before that, gave us his name as "algorithm." There were great books about chess already in the 9th century. The successor to Haroun al-Rashid of the *Thousand and One Nights*, Caliph al-Ma'mun, established a great center of learning and a 25-year flowering of art and science—al-Khwârizmî published there about algebra and arithmetic and geography. One of his books is about the Jewish calendar. I discuss this in *Selected Papers in Computer Science*.

**Q: I understand you are not entirely a partisan of the C++ language.**

C++ has a lot of good features, but it has a lot of dirty corners. If you don't mind those, and you stick to stuff that can be counted well-portable, it's just fine. There are many constructions that are ambiguous, there's no way to parse them and decide what they mean, that you can't trust the compiler to do. For example, you use the "less-than" and "greater-than" signs not only to mean less-than and greater-than but also in templates. There are lots of little things like this, and many things in the implementation, that you can't be sure the compiler will do anything reasonable with.

Languages keep evolving, and that's necessary. I find it impossible to write books for archival without resorting to the English language, though. Whatever computer language is in fashion, you can guarantee that within a decade or two it will be completely

out of fashion. In my books, I try to write things that aren't trendy, but are things that are going to be worth remembering for other generations. I'm trying to distill what, in my best judgment, out of thousands and thousands of things that are coming out now, is most deserving to be remembered.

**Q: You've mentioned Edsgar Dijkstra. What do you think of his work?**

His great strength is that he is uncompromising. It would make him physically ill to think of programming in C++.

**Q: There's a great difference between his scrupulous examination of each and every algorithm, and the practice of commercial programming, where megabytes of code with known and unknown bugs are thrust at the user.**

I know, I know . . . I'm somewhere in the middle. I try to clean up all bugs and I try to write in such a way that, aware of the formal methods, I can be much more sure than I would be without the formal methods. But I don't see any hope of proving everything in these large systems, because the proof would be even more likely to have bugs!

**Q: Programs nowadays depend on huge bodies of code that aren't written by the main author.**

And you look at them and see how each spends most of its time trying to defeat the other. It's all these black boxes you can't open, so you have to build your own firewall.

This is nothing new. In the '60s, someone invented the concept of a "jump trace." This was a way of altering the machine language of a program so it would change the next branch or jump instruction to retain control, so you could execute the program at fairly high speed instead of interpreting each instruction one at a time and record in a file just where a program diverged from sequentiality. By processing this file you could figure out where the program was spending most of its time.

So the first day we had this software running, we applied it to our Fortran compiler supplied by, I suppose it was in those days, Control Data Corporation. We found out it was spending 87 percent of its time reading comments! The reason was that it was translating from one code system into another into another.

**Q: GUIs haven't made this any better.**

We now have so much more processing power that the only people who see what's happening are people writing game programs, who need real high-speed animation. I got new software at Christmas, and I'm really discouraged by the number of failures that I noticed. I'm giving up the idea of using this software to get much done. I'm going to go back and write my books with good old reliable Emacs and TEX.

I still haven't found an editor on the Macintosh where I can create a 1-byte file that has the letter "a" in it that I can send to the PC and read on an Emacs-like editor. I got optical-character-recognition software that has a choice of 50 output formats. Each one of them included all kinds of font-changing mechanisms, and so on. Finally, I found one called "database text" and if I output it in database text, that was what I was expected to get—raw ASCII characters.

It's a natural way to get job security, to make computer systems that need one's expertise.

My TEX system is trying to go the other way, so I wouldn't have to go through the professional typesetters, the professional font designers. I could still use these professionals, but I could use them to provide added value. I didn't have to go through a level of writing something for them to do and then check on it. I can write something, and they can tell me how to improve, but I don't need to write something that they already have on the menu.

**Q: Is the profile of a programmer (which we were discussing earlier) one of an individual who needs this sort of control?**

That's an interesting concept, the need for power! I've always thought of it more in other terms, that the psychological profiling is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large.

When you're writing a program, you're saying, "Add one to the counter," but you know why you're adding one to the counter. You can step back and see a picture of the way a process is moving. Computer scientists see things simultaneously at the low level and the high level.

The other main characteristic of the computer scientist is dealing with nonuniformity of discreet, nonuniform things. We have ten cases instead of one. We don't have one law of thermodynamics, or something. We have case one, case two, case three—each is different. We build models out of nonuniform parts. We're so good at that, we don't see sometimes that a uniform part would work, if it would. But people who are only good at the uniform things could never build a model out of nonuniform parts, could never do the things that a computer scientist does, because they have to find a single rule that's going to cover everything.

We have this aesthetic of cost, how much work it takes to do things. If your mental model is APL, you optimize in different ways than if your mental model is a RISC machine.

**Q: When you look back at the first three volumes of *The Art of Computer Programming* is there anything you would change?**

I have about 900K of changes to the first three volumes, plus changes to other books that I've written, that I plan to make available on my Web page. There are four kinds of changes, and the different kinds are distinguished typographically.

One kind is a "bug" and that means that I have to correct something that is technically wrong. One kind is an "amendment," which means that there is some goodie that deserves to be in there. One kind is an "improvement," something which would go in a future edition of the book, but is probably not worth people's writing it in their own copy. The fourth thing is called a "plan," something still under construction, where the picture is changing so fast that I don't think it's cost efficient for me to write on, since I'll just have to do it again, the kettle is still boiling, but I wish to state that I intend to retool something in a certain way.

It will probably be a while before publishing changes so that entire books can be available online. I don't know how to convert the present system to a better one that will still have the proper incentive structure. There's something all fouled up about the way that software is compensated and font designers are compensated and musicians are compensated, and other intellectual-property rights issues. A scientist can be supported by the National Science Foundation but a font designer is not supported by a National Font Foundation. Both of them are doing things that contribute to the public good.

**Q: Is this just an expression of a love for symmetry, or is there a social injustice being performed here?**

I think that the fact that somebody's expertise is in the field of beauty and graceful strokes and another's is in the field of integrals and differential equations shouldn't mean they have completely different ways of getting paid.

The Free Software Foundation people are putting out good stuff. It's hard for the untrained person to wade through the jargon to install it. Richard Stallman and I don't agree all the way down the line, but he can be an effective arguer! Stallman is one of my heroes, of course. He probably likes some of the things I do, too!

It offends me when people get patents on trivial stuff that we would expect any student to do. I come from a culture where the compensation came because one's work was recognized as advancing civilization. Of course, in literature there were royalties, not

grants. But mostly it was that people had done good work, so you figured they deserved a continuing job. If I were to consider a strategy of becoming rich, it would be so I could support people who need support, who I consider are doing things for the future, like the dukes and duchesses used to support Mozart.

**Q: If you could climb in the pulpit and scold, exhort, and encourage every working programmer in the United States, what would you tell them?**

The first thing I would say is that when you write a program, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. Don't think of it primarily as something a computer is going to follow. The more effective you are at making your program readable, the more effective it's going to be: You'll understand it today, you'll understand it next week, and your successors who are going to maintain and modify it will understand it.

Secondly, ideas that are mathematical in nature should be the property of the world and not of the individual who thinks of the theorem. I'd prefer that all but the most sophisticated algorithms be made public and that everybody use them, and not that every time you use such-and-such a method you should pay a nano-penny to some fund.

I wrote an open letter to the head of the U.S. Patent Commission, published in the current printing of the CWEB manual. I said, "What if lawyers were to have rights to their precedents? What if people had patents on words of the English language, and every author who wanted to write a novel would have to check which words they were using and pay royalties to the owners of those words? Can't you see how obvious it is that the quality of the legal system and the quality of published books would go down? Because you're taking away the building blocks that people need to do their job."

The basic building blocks that software designers need to do their jobs are algorithms and languages and mathematics. It's traditionally impossible to patent a mathematical formula, for very good reason. Anyone who would wish to calculate the area of a circle and use $\pi r^2$ should have to pay a royalty for that: It's exact, it's a universal thing. I think that algorithms should be in exactly the same category. Algorithms are mathematics.

Algorithms are the building blocks to create large, useful systems. The service that you're providing for people is making those systems more accessible, packaging them better, giving better help on the phone, but not just having a method that other people could put into another system.

I would encourage programmers to make their work known the way mathematicians and scientists have done for centuries. It's a comfortable, well-understood system and, you get a lot of satisfaction knowing people like what you did. The whole thing that makes a mathematician's life worthwhile is that he gets the grudging admiration of three or four colleagues.