§10. Excerpts from class, October 19 [notes by TLL]

Should this course have been named "Computer Scientifical Writing" or "Informatical Writing" rather than "Mathematical Writing"? The Computer Science Department is offering this class, but until now we have been talking about topics that are generally of concern to all writers who use mathematics. Today we begin to discuss topics specific to the writing of Computer Science.

We are not abandoning mathematical concerns; Don says that a technical typist in Computer Science must know all that a Math department typist must know plus quite a bit more. He showed us two examples where mathematical journals had trouble presenting programs, algorithms, or concrete mathematics in papers he wrote. In order to solve the first problem, Don had to convince the typesetters at *Acta Arithmetica* to create "floor" and "ceiling" functions by carving off small pieces of the metal type for square brackets. The second problem had to do with typographic conventions for computer programs; *The American Mathematical Monthly* was using different fonts for the same symbol at different points in a procedure, was interchangeably using ":=", ": =", and "=:" to represent an assignment symbol.

Stylistic conventions for programming languages originated with Algol 60. Prior to 1960, FORTRAN and assembly languages were displayed using all uppercase letters in variable-width fonts that did not mix letters and numbers in a pleasant manner. Fortunately, Algol's visual presentation was treated with more care: Myrtle Kellington of ACM worked from the beginning with Peter Naur (editor of the Algol report) to produce a set of conventions concerning, among other things, indentation and the treatment of reserved words.

Don found the prevailing variable-width fonts unacceptable for use in the displayed computer programs in Volume 1 of *The Art of Computer Programming*, and he insisted that he needed fixed-width type. The publishers initially said that it wasn't possible, but they eventually found a way to mix `typewriter` style with roman, **bold**, and *italic*.

Don says he had a difficult time trying to decide how to present algorithms. He could have used a specific programming language, but he was afraid that such a choice would alienate people (either because they hated the language or because they had no access to the language). So he decided to write his algorithms in English.

His Algorithms are presented rather like Theorems with labeled steps; often they have accompanying (but very high-level) flow charts (a technique he first saw in Russian literature of the 1950s). The numbered steps have parenthetical remarks that we would call comments; after 1968 these parenthetical remarks are often invariant relations that can be used in a formal proof of program correctness.

Don has received many letters complimenting him on his approach, but he says it is not really successful. Explaining why, he said, "People keep saying, 'I'm going to present an algorithm in Knuth's style,' and then they completely botch it by ignoring the conventions I think are most important. This style must just be a personal style that works for me. So get a personal style that works for you." In recent papers he has used the pidgin Algol style introduced by Aho, Hopcroft, and Ullman; but he will not change his style for the yet-unfinished volumes of *The Art of Computer Programming* because he wants to keep the entire series consistent.

Don says that a computer program is a piece of literature. ("I look forward to the day when a Pulitzer Prize will be given for the best computer program of the year.") He says that, apart from the benefit to be gained for the readers of our programs, he finds that treating programs in this manner actually helps to make them run smoothly on the computer. ("Because you get it right when you have to think about it that way.")

He gave us a reprint of "Programming Pearls" by Jon Bentley, from *Communications of the ACM* **29** (May 1986), pages 364–369, and told us we had best read it by Wednesday since it will be an important topic of discussion. Don, who was 'guest oyster' for this installment of "Programming Pearls," warned us that "this represents the best thing to come out of the TEX project. If you don't like it, try to conceal your opinions until this course is over."

Bentley published that article only after Don had first published the idea of "literate programming" in the British *Computer Journal*. (Don says that he chose the term in hopes of making the originators of the term "structured programming" feel as guilty when they write illiterate programs as he is made to feel when he writes unstructured programs.) When Bentley wanted to know why Don did not publish this in America, Don said that Americans are illiterate and wouldn't care anyway. Bentley seems to have disagreed with at least part of that statement. (As did many of his readers: The article was so popular that there will now be three columns a year devoted to literate programming.)

As Don began explaining the "`WEB`" system, he restated two previously mentioned principles: The correct way to explain a complex thing is to break it into parts and then explain each part; and things should be explained twice (formally and informally). These two principles lead naturally to programs made up of modules that begin with text (informal explanation) and finish with Pascal (formal explanation).

The `WEB` system allows a programmer to keep one source file that can produce either a typesetting file or a programming language source file, depending on the transforming program used.

Monday's final topic was the "blight on the industry": user manuals. Don would like us to bring in some really stellar examples of bad user manuals. He tried to find some of his favorites but found that they had been improved (or hidden) when he wasn't looking. While he could have brought in the improved manuals, bad examples are much more fun.

He showed a brand-new book, *The AWK Programming Language*, to illustrate a principle often used by the writers of user manuals: Try to write for the absolute novice. He says that many manuals say just that, but then proceed to use jargon that even some experts are uncomfortable with. While the AWK book does not explicitly state this goal, the authors (Aho, Weinberger, Kernighan = AWK) told him that they had this goal in mind.

But the book fails to be comprehensible by novices. It fails because, as Don says, "If you are a person who has been in the field for a long time, you don't realize when you are using jargon." However, Don says that just because the AWK book fails to meet this goal does not mean that it isn't a good book. ("Perhaps the best book in Computer Science published this year.") He explains this by saying, "If you try to write for the novice, you will communicate with the experts—otherwise you communicate with nobody."