Typesetting code listings

Ben Klemens

18 April 2011

Chris Lasher, who is currently reading *Modeling with Data*, wrote to ask me a difficult and delicate question:

> [...] Is there a particular reason why the code listings in the book are in a proportional font, rather than a monospace font? I find myself surprised and frustrated reading the code samples because they're rendered in proportional fonts. It seems even more perplexing given its inconsistency with code snippets, filenames, etc. in-lined with the book's prose, which are typeset in a monospace font.

Since I haven't seen a really thorough discussion of the question, and every book with code has to face up to it, I'll give you everything I've got on the question. There are considerations that advocate for both sides. Just as Chris was frustrated by the variable-width typesetting, some people who looked at the early monospaced drafts made comments about how it was monospace that looked terrible to their eyes.

If you're about to *too long; didn't read* this essay, here's the short version: good code reads like prose, and a variable-width font fits that aim.

• Code should be an essay. The idea of *literate programming* is common enough now that lines about how code should be written for humans first and computers second is kinda cliché, and I wonder if I still need to cite Donald Knuth as a proponent of literate programming or if it's just common wisdom at this point.

Treating code as literature definitely advises that code should be in the same type as any other text. The voice in my head does read code differently when code is in a variable-width font, reading long function names like `apop_beta_from_mean_var` sound more like the pseudo-English they are. This fits with the style I push in the book, where variables (except for the quick throwaways) have full English-word names, and some effort is made to keep one thought per line and one line per thought.

• Letters per line. So I encourage English-word variable names and function names that tell you what they do, and when I wrote Apophenia, I borrowed Mathematica's rule of abbreviating as little as possible, so the reader doesn't have to remember whether to type `apop_est`, `apop_estim`, or `apop_estimate`. That means a lot of characters per line. A variable-width font will put more on a line than a same-hight fixed-width font, because wide characters `[mw-<>]` are much less common than thin characters `[itl().,;/]`. This wasn't an issue for some code, but some snippets would have had every darn line split in two if I did them with a fixed-width font.

That means that a fixed-width font would take up more vertical space on the page, and there'd be less flow with the text. This matters: as the guy who typeset the book, I can tell you that many hours were spent making sure that when the text says *on line six, you can see that...* that you can actually see line six without turning a page.

• Precision. When reading text, you take in the wash of information, and as long as details are unsurprising, your brain passes over them. When writing, you're the one placing all those letters, and need to make sure 100% of them are correct, lest teh reader's subconscious notcie something is wrnog. Because every character gets equal weight, fixed width fonts don't let periods, commas, and parens disappear into the visual background.

So it makes sense that when writing code—or even human-language text—you'd use a fixed-width font. In fact, this makes so much sense that I haven't been able to find any variant of `vi` on my laptop here that would let me write code *without* a fixed width for each letter. Even if I could find such a thing, I probably wouldn't switch.

But when reading, the compiler in your brain will understand what's going on when I write `printf("There are %i errors in this code" 2)` as easily as if I wrote `printf("There are %i errors in this code.", 0);` though if you're C-literate you'll bristle at the missing comma and semicolon as much as you did the English-language typos above. The semicolons and commas absolutely have to be in the right place, but I want you to read the code for meaning, not semicolon placement, and variable-width typefaces take the focus off of punctuation by giving them a fraction of the space.

• The online supplement. You can (and should) get the code online, which gives me a bit of an out: if you hate the typography, you can render it in your favorite font via your text editor. But (and this is less of a cop-out) having two versions of the code changes my expectations. I really did reprint the code in the book so you can read on the bus or at the beach.

[This is something I didn't take lightly. The Apophenia library co-evolved with the book, which means that it was almost by definition going to change post-publication. It did, and each printed code snippet is now a constraint on further evolution. I've put an irrational amount of effort into revising Apophenia in a manner that doesn't break the printed code snippets. My version of Apophenia's test suite, which I run before shipping out a new version, re-tests most of the code snippets in the code supplement. If I'd told you, the reader, that you can't read this book unless you have a PC on hand to view the easy-to-revise online code snippets, my life would have been *much* easier.]

A friend commented to me once or twice that she picks books to read on the bus by how much they'll intimidate other riders (for which *Modeling with Data* is evidently perfect). In my mind, this is how I picture the reader: looking out the window, listening to pop on her headphones, trying to get the big picture, not worrying about where to put the semicolons until she gets to her desk. At her desk, she's got the digital version of the code, which she can inspect character by character in her fixed-width text editor.

• Readers have expectations. Just as I couldn't find a code-oriented text editor on my laptop to render variable width type, I don't know anybody who writes code on the screen with variable-width type. Good typography gets out of the way so the reader can focus on the meaning of the words, not how they look. So the fact that no readers think for a second about code in monospace but some do get thrown by variable-width code is a definite argument for monospace.

• Syntax highlighting (i.e., putting type names and keywords in boldface). Some people put art on the wall beacause they really admire the work and want to have it available to view as much as possible; some people put art on the wall because the wall just looks blank without something.

Code is fundamentally choppy prose. In C, keywords and types tend to appear at the beginning of a line, so putting them in boldface gives a nice cadence. Function declarations alternate type-name, type-name, so putting types in boldface gives a kind of trochaic feel to the line, and gives another visual marker at the head of a new function. For makefiles and the little languages used in the appendix (grep, sed, bash), syntax highlighting felt haphazard and I turned it off.

All of which is to say that I highlight the keywords and types entirely because I like the feel and texture more than without. I've seen a rant or two on this subject by people who are on the other side of the fence—we **don't** boldface verbs when we **write**, do we?—to which I would respond that prose already has cadence, and imposing artificial stresses can only clash with the natural stresses of the language. Though this is obviously a question of æsthetics that has no objective answer, so there's no point in earnestly debating anyway.

When writing code, syntax highlighting is essential, because it is another cue to code that is wrong; at this point I feel a little at sea when I edit code without syntax highlighting. I get the sense that syntax highlighting in books evolved in emulation of the text editors (anybody want to fact check me on this claim?), but I don't care about the history; I just like it.

• Inline code versus code blocks. Writing about code is exceptionally meta-. It's like writing a lingustics book, where we might have a passage about the word *the* versus the word *a*. In fact, I have a TEX macro, `airq`, that I use for words in the text that I'd surround with an air quote and air endquote if I were speaking. And in that sentence you see that I used a monospace type to specify that `airq` is not just a word in the sentence, but the subject of discussion, a sequence of letters that a compiler will parse.

Math books are aware of this, and all mathematical text must be in a separate font from the norm. If you wrote "Let x be a real number," your editor would ding you—it's "let x be a real number." The especially pedantic will point out that using italics is even incorrect, but in practice TEX uses a different non-italic font for math, MSFT Word doesn't really have a different font available, and that determines what you're going to do.

In text, then, I use monospace for everything a parser would look at. This created some really tough calls, by the way, because if the real number x is held in memory as the variable `x`, do I instruct the reader "in the next step, double x" or "next, double `x`"? My case-by-case decisions on this were probably inconsistent.

So code has to be in a monospace font in the text to indicate that it is the subject of discussion, not just a word in my exposition. But none of that is relevant for the code listings in their own blocks. To give a literature metaphor, if I were to compare Poe's opening *During the whole of a dull, dark and soundless day...* to the cliché opening *it was a dark and stormy night*, I would need to use italics to indicate that Poe's text is a subject of discussion. But to print the entirety of the opening sentence, I'd need distinct spacing and a separate paragraph:

> During the whole of a dull, dark, and soundless day in the autumn of the year, when the clouds hung oppressively low in the heavens, I had been passing alone, on horseback, through a singularly dreary tract of country; and at length found myself, as the shades of the evening drew on, within view of the melancholy House of Usher.

By being separated from my exposition about Poe, we recognize all of it as an exhibit of Poe's text, and don't consider it an inconsistency that the block text under discussion has different typography than the in-line. In fact, if I'd followed the typographical convention of italics for the whole pulled segment, you'd get annoyed by the end of it.

Literaure uses a plain font in blocks; block math uses the same math font but more comfortable spacing. So we find that we don't need to rely on a typeface convention to indicate that block text is a special object of study, but could keep the same convention if so desired.