Editor: **Diomidis Spinellis**
Athens University of Economics and Business, dds@aueb.gr

elytS edoC

Diomidis Spinellis

SURE, YOU CAN write English right to left. You can also write software code to look like a disc or even a train (see www.ioccc.org/1988/westley.c and 1986/marshall.c). However, you can't then complain when you have to fight with your magazine's editor or production staff about accepting your column's title for publication, or if your colleagues refuse to touch your code with a 10-foot pole. Writing code in a readable and consistent style is difficult, uninteresting, tedious, underappreciated, and extremely important.

Why

Our code's style encompasses formatting, things like indentation and spacing, commenting, program element order, and identifier names. Although most style choices won't affect the compiled code or the program's runtime behavior, style is a key aspect of the code's maintainability. And because we write code once, but over its life, we read it many times, it pays to keep our code in a style that's easy to analyze, comprehend, review, test, and change.

Expertly styled code is more expressive because, like a master's painting, it communicates at many levels. Indentation and blank lines delineate the code's structure. A well-written comment might indicate how it works. At the statement and expression level, spacing can group together related elements. Aptly named identifiers suggest the meaning of methods and variables, without forcing us to look them up. Similar elements that are placed in an orderly sequence allow us to quickly check for missing entries and insert new ones.

> I've yet to see a top programmer who isn't meticulous about the code's style, or polished code that doesn't form part of a remarkable product.

Many levels of communication engage more parts of our brain, making us more productive. For instance, it might take us a few seconds to parse some deeply nested IF statements, but the pattern-matching part of our brain will instantaneously recognize the nesting through its indentation. Other times, we might visually distinguish the parts of a long, repetitive expression or statement sequence, allowing our mind to tune into the rhythm, follow the logic, and rapidly detect discrepancies. Similarly, when we write some code or design patterns in their commonly used style, our colleagues can immediately recognize their function without wasting time to analyze them. The first time I saw a linked list iteration in C, I rolled my eyes, but now this loop construct in its standard form appears to me just as friendly as Java's iterator. However, any creative deviation from the commonly used style would bring me back to eye rolling, and nobody wants to see that.

The naming of identifiers is another case where programming style can make a big difference. A meaningful name helps you comprehend the identifier's function. If it's accurate, you'll avoid misunderstandings, and if it's also concise, you'll avoid useless typing. If it follows an appropriate naming convention, you'll immediately realize its role (for instance, is it a class or a variable name?). And, most importantly, if all names in a program consistently follow the same conventions, you'll also be able to work in reverse—guess an identifier's name from its perceived function!

In many style issues, style guidelines

...continued on p. 103

0740-7459/11/$26.00 © 2011 IEEE

...continued from p. 104

offer you no choice—you simply follow the rote. Yet, these restrictions are as important as clever creative choices of names and layout schemes. As our poets realized centuries ago, form liberates. Once you start following specific code guidelines, they put on autopilot many trivial but agonizing decisions, such as how to align braces or break up an overly long line. Then, when you write your code, you're not distracted by useless style decisions; when you read it, you aren't put off by inconsistencies or weird formatting choices popping out of the blue. The placement of all the code's nonessential elements, such as spaces in expressions, becomes boringly predictable. Note that meaningless style deviations are simply signals in the code that carry no data. Information theory has a specific term for such signals: noise.

Noise aside, code style is also a powerful signaling mechanism. We humans often jump to conclusions based on proxy signals. We readily associate a bank's granite building with stability and trustworthiness, an aerodynamic car body with a powerful engine, and a person's symmetric features with reproductive fitness. Similarly, when I hunt for bugs, I focus more on the messy code than on the tidy one, and, more often than not, my instinct proves right. This skin-deep impression then moves from the code to the programmer who wrote the software and the organization that sponsored it. Although you might find such conclusions frivolous, let me assure you that, in practice, they work wonders. I've yet to see a top programmer who isn't meticulous about the code's style, or polished code that doesn't form part of a remarkable product.

How

Having seen the many benefits you can derive from improving your code's style, I'm sure you're wondering how to go about it. Rule number one is that you should leave nothing to chance. Place each symbol in your code deliberately, following the code's design, formatting guidelines, and your own conscious decisions. Learn the style guidelines for each language you use (these days, you can't get away with programming in a single language) and apply them religiously. In the beginning, you'll find that consciously thinking about formatting while you code is distracting and slows you down. Change becomes especially hard when breaking your personal habits to follow the established custom. However, you'll see that in a matter of weeks, these changes will become second nature to you, your code will shine, and (on the downside) deviations in other peoples' code will begin to annoy you.

If the style guide leaves something unanswered (say, the way you format a particular SQL extension that your system supports), choose an answer, document it, and then stick to it. A style choice that meshes cleanly with other style decisions is ideal, but don't sweat too much over it. Consistency is more important than particular choices. For the same reason, when editing third-party code, don't impose your own style on it. Derive the style the rest of the code is using from nearby fragments and follow suit. Programming should be an ego-less team activity, where, as in chorus, each of us contributes without being individually heard.

As with writing, you can mightily improve your programming style by reading. Books show you what others have to say on the subject. Look for Brian W. Kernighan and P.J. Plauger's classic, *The Elements of Programming Style*, study chapters 11, 31, and 32 of Steve McConnell's *Code Complete*, and read cover to cover the style guidelines for the languages you use (there are many good online references as well as books). More importantly, read exemplar code written by organizations or individuals you admire. Read their code thoughtfully, focusing just on the style to see how they handle constructs that puzzle you.

Don't rely on code-formatting tools to correct your code after you write it. First, they lack the judgment needed to creatively format the most difficult cases: the ones where good formatting matters most in order to comprehend the code. Second, batch reformatting makes you write messy code, depriving you the advantage of working with code that always shines. Finally, mass reformatting wreaks havoc with version control systems, lumping together functional with formatting changes. Formatting tools are, however, useful when you first clean up your style habits or adopt inconsistently formatted third-party code.

A month ago, a friend bitterly complained about code that looked like a town hit by a tornado. If you write such code, many will be grateful if you mend your ways. And then move on; expert programming entails a lot more than nicely formatted code.

DIOMIDIS SPINELLIS is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of *Code Quality: The Open Source Perspective* (Addison-Wesley, 2006). Contact him at dds@aueb.gr.

Post your comments online by visiting the column's blog:

www.spinellis.gr/tools