DOI:10.1145/1859204.1859219

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

Kode Vicious
Literate Coding

Spelling and grammar do matter.

Dear KV,
Do spelling and punctuation really matter in code? I'm a freshman studying computer science at college, and one of my professors actually takes points off of our programs if we spell things incorrectly or make grammar mistakes in our comments. This is completely unfair. Programmers aren't English majors and we shouldn't be expected to write like them.

Miss Spelled

Dear Miss Spelled,
Who is your professor and can I please send him or her some flowers? Normally I find myself wanting to send people black roses, or a horse's head wrapped in paper, but your letter makes me want to send whoever is teaching you to code a big bouquet of fresh flowers, and a card, and maybe one of those ridiculous balloons that people now send as well.

While it is true that "programmers aren't English majors," there are days—many, many days—that I wish they were, or that they knew one, even tangentially, and offered to help with their science or math homework in return for some help making themselves better understood. The amount of code I have had to beat my head against because the original author did not make his or her intent clear in the comments, when there were comments, is legion, and all that head beating has given me many headaches. It cheers me to know someone else out there is beating your head for me, so maybe, someday, my headaches will go away. I also think the voices might stop at that point, too.

Clear code is actually more aligned with clear writing than most people are willing to admit. Many programmers seem to have gone into their field to avoid subjects such as English or anything that would require communicating with other people rather than machines. That is unfortunate, because code is a form of human communication.

Although a computer will execute your program, it will not execute it in the exact form in which you wrote it, and it definitely does not care that your variables are misspelled or that your comments do not make it clear who did what to whom in the code that follows them. Code is not just for computers; it is for other people as well. If the next person who reads your code—in your case your professor, but after school it will be your workmates—cannot make heads or tails of it because you failed to write in a clear manner, then that is sloppy work on your part and deserves poor marks.

There are many defects that make reading code difficult: from poor structure, to poor formatting, to poor spelling. Attention to detail is the hallmark of all good coders, and that's not just detail in terms of the executing lines of code; it's all of the details, including the words you use and how you use them.

If you are a really poor speller, then run a spell checker over your code before you submit it; and if English is not your first language, then ask one of your classmates to read your comments and tell you if they make sense.

Remember, whether you like them or not, other people are going to read your code, and you owe it to them to make what you wrote easy to read.

KV

Dear KV,

I think I have now come across possibly the most stubborn and recalcitrant programmer I have ever met. He may even rival you in ability to complain about stupid things he encounters. While he's an amazing coder who can produce clear and elegant programs, he has one odd blind spot: he hates debuggers. I don't understand why, and asking him only gets you either a long diatribe or strange mumbling, so I've stopped asking. His preferred form of debugging is via `print` statements, no matter what language he is working in. While I understand that a debugger is not a replacement for writing good code, it's a tool I wouldn't really want to be without. Why would anyone choose `print()` over a debugger?

Bugged and Debugged

Clear code is actually more aligned with clear writing than most people are willing to admit.

Dear Bugged,

I never understand why people who read my columns think of me as stubborn or recalcitrant. Despite my pen name, I am nothing but sweetness and light, and I never complain—I simply point things out.

With that admittedly ridiculous denial out of the way, let's talk about debuggers and people who do and don't use them. Yes, there are people who find debuggers to be an inferior tool and who prefer to use in-program logging, or `printf`, statements to find out where their program is going wrong. I suspect you have also heard about people who cut their own quills and make their own ink for writing.

A debugger is a tool, and if the tool works for you, then you should use it. I see no reason to shun or depend solely on a single tool to help me correct my programs—not that KV ever has bugs in his code.

All that being said, there are many places where a `print` statement is what you actually want. It's important to remember that the debugger interferes with the program it is debugging and can lead to problems with observation. Armchair physicists love to talk about Schrödinger's cat, the thought experiment wherein observing an experiment has the effect of changing the outcome of the experiment. A debugger can have the same effect, and can be far more pernicious than the simple `print` statement. When you're using a debugger on a program, the debugger is introducing overhead, stopping and starting your program using signals, and generally manipulating your program like a sock puppet. While sock puppets are amusing, they're not subtle, and so subtle bugs may not show themselves when you're using the debugger.

A `print` statement, on the other hand, is a simple way to find out if something has gone drastically wrong with your code, and this is why good coders add logging to their systems almost from the start. I've even seen someone use `print` statements to debug multiprocessor locking in an operating system, by printing messages to two different consoles, one tied to each of the CPUs.

The biggest problems I see with using `print` statements for debugging are that they are often left active when they should not be, and in many cases, adding the `print` statements changes the layout of the code in memory. If you're looking for any kind of complex bug—for example, a memory smash, threading, or timing issue—then the humble `print` is more likely to cover up the bug than to help you find it. In these cases, you're better off with a good debugger and hardware that supports watchpoints. A watchpoint is a way to see if a memory location changes at runtime, and it is the best way to find memory smashes in your code—that is, bugs where you're accidentally overwriting memory.

As I'm sure you expected, KV comes down on the side of having many tools—from rough clubs and machetes to finely honed scalpels—for killing bugs.

KV

Related articles on queue.acm.org

A Conversation with Steve Bourne, Eric Allman, and Bryan Cantrill
http://queue.acm.org/detail.cfm?id=1454460

Kode Vicious: Gettin' Your Head Straight
George V. Neville-Neil
http://queue.acm.org/detail.cfm?id=1281885

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and a member of the ACM *Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.