Ode or Code?— Programmers Be Mused!

Stan Kelly-Bootle, Author

My sermon-text this grumpy month is Matt Barton's article "The Fine Art of Computer Programming" (http://www.freewaremagazine.com/articles/focus-software_as_art), in which he extols the virtues of what is widely called *literate programming*. As with the related terms *literary* and *literature*, we have ample room for wranglings of a theological intensity, further amplified by disputes inherent in the questions: "Is computer science or art?" and "What do programmers need to know?" Just as we must prefer *agile* to *clumsy* programming, it's hard to knock anything *literate*. Competing methods tend to sound, like, man, kinda *illiterate*, a term with such a bad reputation that cultures that have not yet invented or borrowed a writing system are called *preliterate*.

Regardless of whoever first coined the intriguing collocation *literate programming*, the seeds were certainly planted by Edsger Wybe Dijkstra at the dawn of serious introspection about the nature of that beast we call programming. The telltale sign that a subject's deepest essence is being introspected seriously is when you encounter the rhetorical format, "What are we *really* doing when we program?" The corresponding question, posed by postmodern LitCritters (mostly Gallic) is: "What are we *really* doing when we write, rewrite, read, and reread?" (The adverb "really" is a warning that some heavy, highbrow brainstorming is pending.)

Dijkstra's oft-quoted advice was, "Besides a mathematical inclination, an exceptionally good mastery of one's native tongue is the most vital asset of a competent programmer."¹ In Dijkstra's other blunt aphorisms, which I always hear in his dry Dutch-accented English monotone, one can also see the seeds of computer-scientific logomachy: the "wars of the programming languages" that still fuel many a curmudgeonly crusade. Poor Basic, exposure to one line of which, Dijkstra claimed, "mentally mutilated potential programmers beyond hope of regeneration."² If that line of code contained the harmful `GOTO`, the reader's fate was instant damnation with a place in the lowest tier of Dante's *Inferno* alongside Judas and the perpetrators of Basic, John Kemeny and Thomas Kurtz (pronounced "calumny and cursed"). Pathetically verbose Cobol, said Dijkstra, "cripples the mind; its

Is your code

LITERATE OR

LITERARY?

teaching should, therefore, be regarded as a criminal offense."³

The fans of C and Pascal sparred in similar fisticuffs,

but perhaps without the ferocity of Dijkstra's "sucker punches." Brian W. Kernighan's "Why Pascal is Not My Favorite Programming Language" (<http://www.lysator.liu.se/c/bwk-on-pascal.html>) remains a classic of reasoned confrontation. Or, rather, as the title hints, less of a direct fight but more of a modest "horses-for-courses"⁴ argument: "Pascal, at least in its standard form, is just plain not suitable for serious programming."

Serious programming, Kernighan explained, is systems programming or anything beyond writing factorial (n) as a student exercise. Pascalians hit back, of course, except for Ould Nick Wirth, Pascal's Onlie Begetter, who maintained a dignified academic aloofness. Owen Hartnett countered Kernighan's taunts in the best possible way: "Owen's Top Ten Reasons Why Pascal is Better than C" uses standup humor to deflate C's proclaimed "seriousness" (<http://www.pascal-central.com/top10.html>).

Owen's fifth reason, for example, reads: "5. In Pascal, when you fool with a pointer or handle, you know you're fooling around with a pointer or a handle. In C, you could be fooling around with anything. C is the ultimate language for computational promiscuity."

This theme of mollycoddling Nanny-Statism (remove every sharp tool from kiddy's sandbox) versus Real-Programmers-Can-Be-Trusted is still a hot button in Java/C++ altercations.

In passing, Owen's second reason, relating to the well-known pun on Niklaus Wirth's name and Pascal's parameter-passing conventions, should have cited the original version we heard directly from the master's lips: "Whereas Europeans generally pronounce my name the right way (Ni-klovs Virt), Americans invariably mangle it into 'Nickle-less Worth.' This is to say that Europeans call me by name, but Americans call me by value." Compare that mock modesty with Owen's more mundane version: "2. Pascal's inventor, Nicholas [sic] Wirth, and parameter-

Continued on page 48

Continued from page 52

passing conventions coalesce to form a nifty pun: You can pronounce his name by reference: Wirth; or by value: Worth. C was invented at Bell Labs where they wouldn't know a joke from a transistor."

I must add, for the sake of that old killjoy called "balance," that C/C++ users seldom take themselves too seriously and openly delight in their obscure and dangerous syntax. I was for many years a judge for the *C Users Journal's* "Obfuscated C" and "C-Pun" contests. It was a matter of perverse pride that there was no "Obfuscated Pascal," "Terse Cobol," or "Transparent APL" contest!

Interestingly, those who attacked language X for lacking structure or objects or components or events or type-safety or exceptions or threads or memory-management, or any of those vogueish foibles-du-jour that spell "seriousness," soon found X reincarnated with those very assets variously attached. But, always to remember: the features must accrete *seamlessly*. Be like Hamlet: "Seams, madam! Nay, I know not 'seams.'" (act 1, scene 2) The merest sign of seaminess, and your opponents will claim "afterthought kludge." Thus, we find Basic forever extended (as in "extended family," though some say as in, "My resources are rather extended").

To bowdlerize Professor Peter Fellgett (coinventor of Ambisonics): "Basic is manure. Extended Basic is manure with icing."⁵ Those layers of "icing," one could argue, have now almost obscured the midden below. Visual Basic .NET is more than a language; it's a whole bleeding way of life—components rising through the floors and dripping from the rafters—the shamelessly preferred platform for business programmers. Sixty-two percent of developers use some form of VB by Microsoft's reckoning, vying for third place with C++ and JavaScript, behind the top two, C# and Java. If Dijkstra were alive today, he would be turning in his grave (© Joe Miller).

The language wars are complicated by the many ways languages evolve. Lessons *are* learned. Past "errors" are pardoned as inevitable steps leading to nirvana. Honest. Wirth's own Modula-2 is more than an extended Pascal. C++ is now much more than a better C, a C plus Simula, or a C with classes. Recall the joke, openly relished by Bjarne Stroustrup Himself, that ++C (pre-increment [improve] C, then use [return the value of] the improved version) might have been a better name syntax-wise than Rick Mascitti's suggested C++ (use [return the value of] the old C, then post-increment [improve] it!). Such tales have been further embellished by arcane suggestions that the operator ++ can be suitably overloaded for objects of the class Language! I offer one small observation in favor

of the name C++ over ++C: googling C++ wins you 6 million++ hits, but googling ++C is a disaster since the initial ++ is ignored, revealing all those K&R C sites: you don't get no stinking ++C matches (© John Huston).

Whatever the origins of literate programming, there's no doubt that its fame and/or infame⁶ comes from the great King Knuth. For 'twas he, the noble El Don, who first propounded his version of the "literate" approach to coding in that spooky, fatidic year 1984.⁷ His ideas were later amplified and published in 1992 (*Literate Programming*. Lecture notes, Center for the Study of Language and Information, Stanford). Those who in-joke about the publicational time gap between volumes 3 and 4 of Knuth's magnum opus, TAOCP (The Art of Computer Programming), should remember this and all the other fine work that has distracted him, especially his Herculean efforts in typesetting and typographical computing.

Barton's plea under the mantras, "Code isn't just for computers" and "Reading programs for pleasure," is to promote code that humans can enjoy reading for the sheer fun of it, in the same way, for example, that they can enjoy curling up in bed with their favorite Trollope (an author carefully chosen for a cheap thrill unworthy of this august journal). We note first a possible confusion or overlap between *literate* and *literary* programming. Dijkstra tends to stress literacy in the sense of fluent command of one's working/publishing tongue (and that really means English for most practical purposes), so that all text not directly compilable, such as comments and explanations, would be written crisply and free from ambiguities. Barton seems to be seeking a literary flair in the code itself.

We therefore note next that Barton's plea applies chiefly to reading the *source* code of a reasonably high-level language, and one in which the human reader is reasonably fluent. The meaning of "reasonably high-level" is, of course, user-defined, insofar as some of my most elegant code was written for 68K assemblers—and still a joy to read, if I'm allowed an immodest moment, after several decades of rusty absence.⁸ Whether you might gain pleasure from reading binaries is moot only if you are particularly devoted and unnaturally perceptive. I've no doubt that a Turing, Knuth, or Toxen would nod with delight at long strings of 0s and 1s: "Gee, grab that 0011001100 instruction! How Nabokovian! Such precision! And a palindrome to boot—nay, to kill for!" (Coincidentally, my column concludes with an elegant test for palindromicity.) For the rest of us, painful memories of poring over core dumps would be a major deterrent.

So, we are looking for code-text at least halfway legible to scan and enjoy as literature. Alas (but only in the sense

of confounding our judgments), literature comes in all shades of artfulness and comprehensibility, and there's no ready correlation between greatness and reader friendliness. Indeed, compare the undoubted, widely acclaimed classics: Jane Austen's *Pride and Prejudice* and James Joyce's *Finnegans Wake*. What might be the programming analogs of such diverse masterpieces? Austen's clear, lucid style could possibly be compared with Modula-2, while Joyce reminds us again of those obfuscated-C teasers or a brilliant set of sequences in APL⁹—arcane but well worth the effort to untangle.

In our list-mad and lit-prize market-driven society, we tend to delegate our literary judgments to the “experts,” or at least to be strongly influenced by them. By buying their choices, we make the best-seller tables into self-fulfilling prophecies. J. Peder Zane's largest-ever “ten greatest books of all time”¹⁰ survey has just concluded by combining the votes of 125 authors considered (by whom?) to be the “125 greatest living authors.” Lev Grossman and I agree that “literary lists are basically an obscenity. Literature is the realm of the ineffable and the unquantifiable” (*Time*, Jan. 15, 2007). Yet the scale of the project does make it “a pretty interesting obscenity.”¹¹

One obvious flaw is that writers over the millennia have been uncritically bitchy about their competitors

(try googling “literary feuds”). Robert Greene jealously attacked a more successful fellow playwright as “...an upstart crow...in his owne conceit the onely Shake-scene in a countrey.” That was 1592, and William Shakespeare was not amused. In the 1960s, Vladimir Nabokov's *Strong Opinions* were very strong indeed on such “frauds” as Cervantes, Dostoevsky, Pasternak, and Orwell. Is Vlad the Impaler vindicated, I wonder, by the fact (surprising many Nabokovians) that his *Lolita* came in fourth in the ultimate, overall “Best Books of All Time” list, whereas nothing by his four derided targets made the top 10. Nabokov never cared much for prizes or others' opinions, yet he would relish the fact that the three books ahead of *Lolita* were by his favorite authors (Tolstoy in first and third places with *Anna Karenina* and *War and Peace*, and Flaubert in second place with *Madame Bovary*). When you ponder that Mark Twain's *Huckleberry Finn*, at fifth, was ahead of Bill the Quill's *Hamlet* at sixth, you do begin to doubt the sanity of numerically ordering the “better-than” relationship for works of art. Statisticians will be particularly aware of the distortions arising when you combine lists of preferences. Overloading “>” to mean “voter preference,” it's well known that transitivity can be lost. Thus $a > b$ and $b > c$ does not always imply $a > c$.

Comparing programs is only a teeny-tiny bit easier!

Regardless of source-code legibility (watch those indents, you sloppy wretch) and choice of object hierarchies, we do have the compiler and stopwatch as objective metrics for accuracy (bug count) and efficiency (runtime). To be up-front, hard-nosed, blue-boiler-suit about it: “Sod the code layout, mate, where’s our payroll?”

Back comes the cry: “But debugging and maintenance demand code legibility.” Here follows a bifurcation in the literate programming route. Ray Giguette sees a helpful literary role right at the start of the project, using literary analogies to shape our approach to software design.¹² Robert McGrath dismisses this too brusquely, I believe, while admitting that even weak analogies may help to improve understanding between humans involved in design and coding.¹³

Meanwhile, Barton, with more modest goals, refers to an emerging “band of literary critics of code,” ready to guide the growing “appreciative readership for source code.” Leading this band is Diomidis Spinellis, author of *Reading Code: the Open Source Perspective* (Addison Wesley, 2003). Yet much of Spinellis’s *Reading Code* is devoted to what I call politely the preliterate phase of programming. The emphasis is on reading code not for “literary plea-

sure” but for reasons that have always been high priorities: (1) to understand the intentions of the programmer; and (2) to debug or improve the code.

There are, surely, doryphoric pleasures aplenty in exposing others’ howlers (the origin of the triumphal “Yahoo!”, I believe). The more the layout is pleasant and logical, the less boring are the debugging sessions. And we’re now beginning to see various threads converge to add dynamic interactions that will help you make sense of any screen display: Ted Nelson’s hyperlinks and Tim Berner-Lee’s HTML. It’s called *Charlotte’s Web*, I think. For program text, the virtual function names are fish-belly gray; missing semicolons flash in false-azure. You can click on bastard objects to locate their putative parents.

Literature is closing in, too. A particularly spectacular example is ADAonline at <http://www.ada.auckland.ac.nz/index.htm>, giving the text of Nabokov’s vast, allusion-rich novel *Ada or Ardor: A Family Chronicle*, which really belongs in the top 10. Brian Boyd provides clickable links to explain the allusions.

How can we ever forget that all human discourse ultimately rests on NL (natural language)? I’m telling you this in NL, not computer code. Peter Naur tells it right:

acm
queue
architecting tomorrow's computing

**Coming Soon
in Queue**

Commodity Enterprise Middleware

The Seven Deadly Sins of Linux Security

A Conversation with Michael Stonebraker

he thanks ACM for his 2005 Turing Award but reminds us that human thinking is beyond Turing machines.¹⁴ Science is about descriptions. These may end up using precise, consistent, convenient symbols, but first we need to chat (maybe banter!) about our goals, moving with practice from informal to less informal discourse. The more types, registers, styles of NL we've been exposed to, the better. The old whinge that NL is saddled with innate ambiguities need not paralyze us. Sipping from the pools of great literature alerts us to the crafty ways of exploiting ambiguity. Will you BackUs or IgNaurus?

Finally, as promised, here is an example from Bjarne Stroustrup of a function returning true if presented with a palindromic string. Read and enjoy the recursion! ☺

```
bool is_palindrome(const char* first, const char* last)
// first points to the first letter, last to the last letter
{
if (first<last) {
    if (*first!=*last) return false;
    return is_palindrome(++first,--last);
}
return true;
}
```

REFERENCES

1. Dijkstra, E.W. 1975. How do we tell truths that might hurt? <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD498.html>.
2. *ibid.*
3. *ibid.*
4. The frisky “horses-for-courses” idiom may not have galloped beyond Albion’s shores. Even Brits use it without due process. It simply states the obvious: that objects may be differentiated ontologically by their innate purposefulness. Easy for me to say. Liverpool Dock Road cart horses do not win the Pimlico Special. Seabiscuit never hauled a ton of stolen coal. Pascal, some claimed, was a “teaching” language. C, others claimed, was “unteachable,” acquired through trial and error, chiefly error.
5. Kelly-Bootle, S. 1995. *The Computer Contradictionary*. Cambridge: The MIT Press. My entry at Extended Basic gives the unexpurgated version.
6. A clever, illiterarily back-formed noun from the more familiar adjective *infamous*. This reminds me that my current bête noire is the overuse of *famously*, as in “As Churchill famously remarked...”
7. In keeping with my subject matter, *fatidic* is a more elitist, literary choice than *prophetic*. Was Orwell’s pes-
8. Kelly-Bootle, S. 1988. *680x0 Programming by Example*. Indianapolis: Howard W. Sams & Company. I find that I wrote in the introduction, “...one of the objects of the book is to improve your reading skills!”
9. Well-versed readers will know that *Finnegans Wake* is fond of an APL anagram: ALP crops up as Anna Livia Plurabelle. LitCritters get quite excited over such minutiae. I must confess my analogy is imperfect. APLers will rush to tell me that the garrulous, circular *Finnegans Wake* (it opens with “riverun” and ends with “the”—a sort of GOTO START endless loop) can be expressed in just three lines. Dijkstra’s spin was quite acerbic: “APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.”
10. Peder Zane, J. 2007. *The Top 10: Writers Pick Their Favorite Books*. New York: W.W. Norton.
11. *ibid.*
12. Giguette, R. 2006. Building objects out of Plato: Applying philosophy, symbolism, and analogy to software design. *Communications of the ACM* 49(10): 66-71.
13. McGrath, R.E. 2007. Programs are not literature, even by analogy. *Communications of the ACM* 50(1): 11.
14. Naur, P. Computing versus human thinking. 2007. *Communications of the ACM* 50(1): 85-94. Well worth the reading effort. Improve your command of NL and your ability to detect and possibly ignore the odd whiffs of justified, cranky victimhood.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

STAN KELLY-BOOTLE (<http://www.feniks.com/skb/>; <http://www.sarcheck.com>), born in Liverpool, England, read pure mathematics at Cambridge in the 1950s before tackling the impurities of computer science on the pioneering EDSAC I. His many books include *The Devil’s DP Dictionary* (McGraw-Hill, 1981), *Understanding Unix* (Sybex, 1994), and the recent e-book *Computer Language—The Stan Kelly-Bootle Reader* (<http://tinyurl.com/ab68>). *Software Development Magazine* has named him as the first recipient of the new annual Stan Kelly-Bootle Eclectech Award for his “lifetime achievements in technology and letters.” Neither Nobel nor Turing achieved such prized eponymous recognition. Under his nom-de-folk, Stan Kelly, he has enjoyed a parallel career as a singer and songwriter.

© 2007 ACM 1542-7730/07/0400 \$5.00