But, Having Said That, ...

Stan Kelly-Bootle, Author

A persistent rule of thumb in the programming trade is the 80/20 rule: “80 percent of the useful work is performed by 20 percent of the code.” As with gas mileage, your performance statistics may vary, and given the mensurational vagaries of body parts such as thumbs (unless you take the French *pouce* as an exact nonmetric inch), you may prefer a 90/10 partition of labor. With some of the bloated code-generating meta-frameworks floating around, cynics have suggested a 99/1 rule—if you can locate that frantic 1 percent. Whatever the ratio, the concept has proved useful in performance tuning.

An intriguing analogy exists in NL (natural language). In English, for example, a reasonably exact 99/1 split has been found by counting those entries in the OED (*Oxford English Dictionary*) that have Anglo-Saxon roots. You will be aware that English is (in)famous for borrowing outrageously from other tongues, but the extent of the borrowing may surprise you. Just 1 percent of our current lexicon is truly “native” Old English from around 500-700 CE, and even that language was a complex mix of marauding Old Norse and other Germanics, with a colorful dash of indigenous Celtic. Yet, the further startling fact is that the 1 percent of natively derived words makes up some 60 percent of everyday usage. The reason, you’ve probably guessed, is that most of those short, busy words such as *the*, *a/an*, *I/me*, *if*, *and*, *but*, and *not* are of Anglo-Saxon origin in addition to the many common nouns such as *axe*, *shield*, and *blood* that enliven *Beowulf*!

The borrowed and assimilated words that make up the other 99 percent in the OED reflect the major influences of Norman-French and Latin, swollen by the large number of artificial “inkhorn” terms noted and mocked by Samuel Johnson. These were coined from classical Greek and Latin roots by elitist scholars, disdaining the lowly peasant Anglo-Saxon. Thus, the rich *navigated* while the poor merely *sailed*. I use the term *borrowed* loosely. The process of linguistic evolution through contact and intermarriage is much more complicated and involves subtle grammatical exchanges, as well as sharing vocabulary.

Can we transfer the benefits of the programming division-of-labor rule to NL discourse? If we can better tweak system performance by finding and massaging the

**All ifs, ANDs,
AND (ESPECIALLY)
BUTs.**

more productive routines, should we not pay more attention to that hyperactive 1 percent of our lexis that dominates our word-

frequency charts?

I remind you of two motivations. First, Dijkstra’s plea that NL mastery should precede your programming-language lessons. Second, Jef Raskin’s thesis that “comments are more important than code.”¹ Although somewhat Raskinesque tongue-in-cheek, his message is really this: “The thorough use of internal documentation is one of the most overlooked ways of improving software quality and speeding implementation.” This statement is hard to falsify and carries enough bland weaseling to recall Peter Fellgett’s “Cook until done” generic menu: “Into a clean dish, place the dry ingredients and add the liquids until the right consistency is obtained. Turn out into suitable containers and cook until done...”²

The IT equivalents are “Press the appropriate key,” or “Assemble the best possible group of programmers for the task at hand.”

My own contribution to the controversy is YACC (Yet Another Comment Compiler),³ an extremely eXtreme solution that ignores your code and compiles your comments. In pass 1, for example, YACC converts

```
i++; // post-increment counter by 1
```

```
to
```

```
post-increment counter by 1 // i++;
```

Pass 2 is still a work in progress.

The basic words essential to logical discourse (hence, to writing code) such as *if*, *and*, *or*, and *not* are indeed Anglo-Saxon and have been subject to centuries of exegesis. The word I propose for overdue study is *but*. This innocent-looking token is chiefly used as a conjunction, separating two contrasting clauses: “I love Algol but hate Fortran.” This construction is surprisingly prevalent,

Continued on page 62

Continued from page 64

sometimes hiding under the near-cognates *yet* and *however*. But,⁴ *but* can also act as a preposition meaning *except* or *excluding*. Thus, “I love all languages but Fortran.”

Sensible uses of *but* should exclude the conjunction of unrelated clauses, as in “I love chocolate but hate Fortran.” Occasionally, on closer parsing, you may find a sentence asserting “p but not-p,” which is rather like hedging your bets by answering a question with “yes and no.” One can often defend this in NL rhetoric where the propositions may lack binary truth-values (“Fortran is bad!”), but elsewhere one cannot escape the rigors of the Law of the Excluded Middle: Only one of p and not-p can be true (with the other false), even if some logics allow the impossibility of being able to prove which is which.⁵

All of which may explain why *but* is helpful in your comments but absent as a logical keyword in your code. A rare exception is my own LEGOL language,⁶ which uses a variant of `#include` directive:

```
#including_but_not_limited_to
```

What LEGOL lacks in terseness, it makes up for in a precision that reflects centuries of painful jurisprudence. Thus, the sloppy declaration

```
int i = 1;
```

is rendered as

Be It Understood And Acknowledged By These Presents That: the newly created object herewith to be named and referred to as i within the scope determined by the preferred embodiment of the previously submitted namespace patent hereby incorporated by reference and further that i being of the type declared known and widely recognized as integer it shall straightway without let or hindrance be assigned and allotted ne quid nimis the value 1 (ONE) and shall retain that value sub tegmine fagi until such time or times if any that some other value within the jurisdiction of the Type Safety (Promotions) Act may be assigned and allotted thereto.

You may also argue that XOR carries the soul of a *but*, as in “A XOR B,” meaning “A OR B BUT not BOTH.” This *but* is really an *and*, reminding me of the poor chap whose car was stolen but his insurance claim rejected. The policy covered fire AND theft, and fire could not be proved. My own favorite Boolean quiddity is the A XAND B, defined as “A AND B but not BOTH.”

Finally, I must thank the many readers who responded to my column, “Anything Su Doku I Can Do Better” (*ACM Queue*, December/January 2005/2006). Programs to solve the puzzle were provided by Paul Eggert (GNU Prolog), Peter Klammer (VB6), and Marc Auslander (C++). Comments arrived from Bjarne Stroustrup, and others pointed to Donald Knuth’s contributions to the problem. There’s no doubt that brute-force backtracking can in theory solve all Su Doku puzzles. The “well-formed” puzzles appearing in newspapers have 25 or more clues and a unique solution, which the readers’ programs solve quickly. The open CS question, which drags us into the P-NP jungle, is how to handle the “ill-formed” puzzles. Give Eggert’s program an empty grid (no clues) and it will generate all 10^{21} (approx!) answers, providing the universe runs long enough and the paper supply holds up. ☺

REFERENCES

1. Raskin, J. 2005. Comments are more important than code. *ACM Queue* 3(2): 64.
2. Fellgett, P. B. 1988. Cybernetics and roux sauce. In *But the Crackling is Superb, An Anthology on Food and Drink by Fellows and Foreign Members of The Royal Society*, ed. N. and G. Kurti. Bristol, UK: Adam Higer.
3. Kelly-Bootle, S. 1995. *The Computer Contradictionary*. MIT Press.
4. Ignore the pop grammarians who denounce sentences starting with *And* or *But*. See Bryson, B. 2002. *Troublesome Words*. Penguin Books.
5. For some Eastern doubts on this Western “desicating and imperialistic linear” logic, see Faure, B. 2004. *Double Exposure*. Stanford, CA: Stanford University Press.
6. See reference 3.

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) and *Understanding Unix* (Sybex, 1994). *Software Development Magazine* named him the first recipient of the annual Stan Kelly-Bootle ElecTech 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.

© 2006 ACM 1542-7730/06/0300 \$5.00