

# Re: Code Comprehension

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.programming/2006-08/msg01061.html>

---

- *From:* Logan Shaw <lshaw-usenet@xxxxxxxxxxxxxx>
  - *Date:* Tue, 29 Aug 2006 02:07:38 GMT
- 

Chris Uppal wrote:

Not really following on from that, but this is a convenient point to raise an issue which interests me, and which we seem to be skirting around without ever quite touching explicitly.

Given a language like C, where the actual semantics differ quite considerably from what I would describe (with no offense intended) as "what a PASCAL programmer would expect"[\*] and further assuming that the programmer is intent on making the program text as clear and readable as possible (not necessarily compatible aims), what is the best way to use that language's semantics ?

It seems to me that there are two options, or at least two ends to a spectrum. At one end, you can program to the languages /actual/ semantics, expecting the reader to understand the language itself (not some idealised approximation), and making use of idioms which depend on that. At the other end, you can construct a subset of the language (which approximates as closely as you can manage to the "ideal" version), and then stick rigidly to programming in that subset.

I think that's a good point. And I think the answer to where to land on that continuum depends on your level of familiarity with the language.

To me, two of the most important things that help achieve clarity are (1) being explicit (making your intentions known and being sure to include little details that steer people in the right direction), and (2) removing clutter that acts a distraction from more important things.

Obviously, these two are at odds with each other sometimes. One man's helpful detail is another man's clutter.

It's like working at a new company vs. working at a company you've been with for years. When you're new, you prefer if everyone speaks in sentences like, "Bob, the project manager, asked Karen, the receptionist, who also does other clerical work, to mail the package

## Re: Code Comprehension

to Acme, Inc., who is our biggest customer." If you've been there 10 years, you'd rather hear, "Karen is mailing an Acme package for Bob." The things that are unclear to the newcomer are obvious to you because of your level of familiarity. If they are stated, they are just noise.

So the answer to this question, with regard to a particular programming project in a particular language, is that you ought to write code depending on what level of familiarity the present (and future) team has with the language. You could argue that you should shoot for code being as non-idiomatic as possible to be understandable to the widest range of programmers, but I would argue that, generally, someone who isn't familiar with even the basic idioms of the language probably is not ready to work on the project.

But it's not that simple. Some language features are a pure, unadulterated mistake and are to be avoided. Some are tempting shortcuts that reduce code quality. Judging which features are worthwhile and which aren't is not a simple thing.

It is my no means obvious that it is a good thing to "pretend" that a language's semantics are other than they are. My opinion is that it is bad — if nothing else it gives the reader two levels of semantics to think about: what does the program actually mean (given the real semantics) and what does the writer /think/ it means (using whatever, unspecified, semantics s/he has invented as an "ideal").

Another poster raised a good question: if you have a set of idealized semantics, how do you know that you are writing correct code? One answer is that there are idealized sets of semantics that lead to producing correct code and other sets of semantics that don't.

To give a trivial example, if your idealized set of semantics is exactly the same as the true semantics of C except that you have erased from your mind the existence of the "~" (bitwise negation) operator, you can be confident your mental model will lead you produce correct code. On the other hand, if your mental model says that "&&" is the logical "and" operator but doesn't account for the fact that it has lazy evaluation and instead assumes that the entire expression is always evaluated, you might find yourself writing incorrect code.

As a practical matter, if you are going to come up with an idealized set of semantics, it helps to know the real semantics very thoroughly. And that could be a worthwhile endeavor. However, it would not be a worthwhile endeavor to attempt to produce an improved set of semantics without a sufficient understanding to do it correctly.

## Re: Code Comprehension

It is true that there can be idioms which lend themselves to misinterpretation, but they tend to be rare — idioms become idioms because (considered as memes) they are /successful/.

Often, but it's not always the case that a successful meme or idiom is a helpful one, so that has to be taken into account.

– Logan

.