

# Re: Debugging standard C library routines

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2006-10/msg00460.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-10/msg00460.html)

---

- *From:* Richard Heathfield <[invalid@xxxxxxxxxxxxxxxxxx](mailto:invalid@xxxxxxxxxxxxxxxxxx)>
  - *Date:* Sun, 01 Oct 2006 20:52:38 +0000
- 

Eric Sosman said:

This programmer you describe seems a bit of an odd fish. He's reckless enough to ignore warning messages, yet diligent enough to initialize all his pointers. Sounds like a person with a multiple personality disorder ...

Well, of course what we're really talking about is "what constitutes good practice?" What kind of code is easiest to debug? What kind of strategy works best for dealing with diagnostic messages? Do we strive for a clean compile at all costs? I used to do so, but I found that sometimes I couldn't /get/ a clean compile without adding spurious casts to the code, which I was loathe to do. So we have to consider a strategy on how to deal with warnings. When is a warning significant? To me, the answer is "it's significant for as long as you can't understand why it's there". So – if a compiler told me "hey, you might be using this thing before giving it a proper value", then yes, I'd take it seriously. Unfortunately, it is not a panacea (and yet it certainly looks like one). It can't, for example, deal with `foo(&obj)`; – and so now you have the worst of both worlds – no diagnostic message AND no initialisation. Oopsie.

The cheapest errors are those not made in the first place.  
The next-cheapest are those caught by the compiler and fixed  
before committing the code.

That's fine, provided people treat diagnostic messages seriously. We have ample evidence here on `comp.lang.c` that this is not the case.

It's interesting to read this in light of remarks about "drool-proof languages" on another current thread ...

Yes, I'm trying hard to avoid quoting Sturgeon's Law here...

## Re: Debugging standard C library routines

I feel -- without quantitative evidence, I admit -- that there's more to be gained from improving a programmer's skills than by trying to compensate for his deficiencies.

I'm sticking my neck out here, but I am willing to bet any amount up to and including a penny that you spend hardly any time debugging your own code. Not \*no\* time, but hardly any time. I suspect that you make a low number of mistakes, and that most of those you do make are fixed within seconds or, at most, minutes of your seeing their symptoms emerge for the first time. It's certainly true for me, anyway, and I'd be surprised if it weren't true for you too. And the reason is obvious -- you know what you're doing, and you've evolved a strategy for successful code development that \*works\* -- for you. And I've evolved a strategy that works for me. There's nothing in the rules says we have to have evolved the /same/ strategy. And maybe that's why we're arguing. :-)

Now, when I have to debug someone else's code, that can take a bit longer, because they didn't write it the way I would have wanted them to write it. When debugging other people's code, then, and given the choice, I'd rather chase a stable target than a Heisenbug. So I recommend to people that they write the code in a way that makes it easy for /me/ to debug it. (After all, that's probably what I'll end up doing anyway!)

Wanton initialization of pointers (of any variables, actually) discourages the compiler's assistance and therefore ought not to be indulged in.

Giving the program deterministic behaviour by ensuring that all variables are initialised helps the programmer to understand the program better and debug it more quickly, and therefore ought to be encouraged. :-)

Get rid of the STOP sign at the busy intersection, but make sure that anyone who fails to stop \*will\* be run over by a cement truck.

Yes, absolutely. In a computer, nobody gets hurt (at least, not during initial program development!) -- we can run the little variables up to the stop sign over and over again, and watch them go over the line and get crushed by the truck over and over, and eventually we figure there's something wrong with their brakes. So we go fix that. Typically, it takes a few seconds, half of which are spent recovering from slapping our own heads -- "DUH!" -- as we marvel at how we could have been so dumb.

Re: Debugging standard C library routines

---

Richard Heathfield

"Usenet is a strange place" – dmr 29/7/1999

<http://www.cpax.org.uk>

email: rjh at above domain (but drop the www, obviously)

.