

Re: Iteration in lisp

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2008-04/msg01675.html>

- *From:* Duane Rettig <duane@xxxxxxxx>
 - *Date:* Wed, 30 Apr 2008 00:15:23 -0700
-

Scott Burson <FSet.SLB@xxxxxxxx> writes:

On Apr 26, 8:34 pm, Kent M Pitman <pit...@xxxxxxxxxxxx> wrote:

[I have to admit that my first reading of Kent's articles on this subject came from a slightly different point of view, (since Allegro CL indeed never warns when you simply `setq` a variable at the repl) but because of this follow-on I have to speak up for the implementors, perhaps speaking indirectly to Kent as well – it really depends on how extensive his use of "ever" in this paragraph was meant to be...]

I also think the inability to do

```
(setq x 1)
```

in a textbook is a travesty because it forces people to get into the discussion of `DEFVAR` way too early. You want to be able to talk about and test program fragments, and the inability to do that "portably" is silly. I consider it unreasonable that an implementation ever complain about this.

Hear, hear!

I don't understand how some implementors have convinced themselves that emitting warnings in this case is acceptable. Yes, I know that the standard doesn't really say what should happen and this is technically an error. But it doesn't make sense to elevate the standard above the precedent that has been set by existing implementations when that precedent is important to users.

As I said in the bracketed comments above, I originally didn't read much into this issue because Allegro CL doesn't warn you when you've assigned to an undefined variable at the repl, since it is likely that the person doing the typing is looking for somewhat of a "dwim"

Re: Iteration in lisp

experience, and we believe that warning in that case is not too helpful. (I always got a kick out of anal programs, in any language, that had error messages that say something like "Error: missing comma after 'this'" [well, why doesn't the program give the programmer a break and add the comma – if it's unambiguous, what's the harm? that the wayward programmer will never learn his lesson??] ... sheesh)

I think that it is a more likely correlation that those lisps that don't have an interpreter, or lisps which by default are set to compile their forms, will be the ones which end up warning in this way.

However, there is absolutely a very good reason for doing this warning in other situations – namely, in compiled code. That reason is, in a single word: "spelling". In reality, lisp programs which are candidates for compilation are less likely to have `(setq x 1)` for a form, and more likely to have a form that looks something like `(setq *long-variable-with-spelling-that-doesnt-match-its-defunition* 1)` for which it may be hard to see the mistake (how long did it take for you to find it?). In this case it is likely that no actual variable looks like this one, and thus it is likely that this name is a mistake. And for both categories of programmers: those of us whose typing is painfully slow, and for those who type like lightning, the result is always the same: some words inevitably end up misspelled. Hence the reasoning for a warning.

The standard is supposed to serve the user community. Some seem to think it should be the other way around. I think this is the fundamental problem.

That's precisely what we try to do. Some things we do have that effect better than others. You can bet that most of the behaviors in most of the CL implementations have been considered very carefully, and to the extent that the behaviors are different, well, it's simply due to different resting points on many tradeoff axes. The question of whether to compile top-level forms automatically or to require the user to explicitly call `compile` on them is a good example. There are reasons on both sides of the argument, and some prefer one way and others prefer the other way. Some consequences come more or less directly out of such decisions, like the behavior of warnings on `setqs` of undefined variables. I think your statement is much too harsh. You should instead strive to understand `_why_` the decisions are made the way they are, and that will bring enlightenment through many facets. Many implementors are very accessible and can shed light on some of these tradeoffs, some on this very newsgroup.

It really astounded me the unsympathetic response I got the last time I raised this issue here. Nice to know you're on my side, at least :)

Re: Iteration in lisp

I'd be interested to read that thread (hopefully I wasn't a contributor to the non-sympathy, though I won't guarantee I will agree with all of your arguments). Do you have a pointer to that thread?

—
Duane Rettig duane@xxxxxxxxx Franz Inc. <http://www.franz.com/>
555 12th St., Suite 1450 <http://www.555citycenter.com/>
Oakland, Ca. 94607 Phone: (510) 452-2000; Fax: (510) 452-0182
.