

Re: the sort function in lisp (destructive)

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2008-01/msg00658.html>

- *From:* rem642b@xxxxxxxxxx (Robert Maas, see <http://tinyurl.com/uh3t>)
 - *Date:* Fri, 11 Jan 2008 11:47:19 -0800
-

From: Madhu <enom...@xxxxxxxx>

As for SORT Robert Maas posted a small hack that would preserve the head cons after calling the implementation's SORT in SORT-KEEPING-HEAD-CELL in his post[1] <News:rem-2007oct11-...@xxxxxxxx>

Dated "11 Oct 2007 02:44:41 -0700"]

Note I haven't fully tested that code, so corrections would be welcome.

But it's nice that I'm fondly remembered for my occasional hackery.

<<http://groups.google.com/group/comp.lang.lisp/msg/c99a0c00771261bc?dmode=source>>

However I think I have a cleaner idea: Keep aside a single CONS cell (or re-allocate it each time for better thread safety but increased CONSing), temporarily splice it in place of the user's head cell before sorting, then splice the user's head cell back in after sorting. No special cases are needed except for empty (or single-element) list.

```
(defvar *temphead* (cons nil nil))
(defun sort-keeping-head-cell (userhead sortfn)
  (when (cdr userhead)
    (shiftf (car *temphead*) (car userhead) nil)
    (shiftf (cdr *temphead*) (cdr userhead) nil)
    (setq *temphead* (sort *temphead* sortfn))
    (shiftf (car userhead) (car *temphead*) nil)
    (shiftf (cdr userhead) (cdr *temphead*) nil)
  )
  userhead)
(setq l1 (list 1 3 5 2 4 6))
--> (1 3 5 2 4 6)
(sort-keeping-head-cell l1 #'>)
--> (6 5 4 3 2 1)
l1
--> (6 5 4 3 2 1)
```

Of course, with either original (ugly) or new (clean) version, you really want to mess with &rest argument to have full flexibility of the built-in SORT function's keyword arguments.

Aside lecture to newbies: In Lisp, we often pass a pointer to a

Re: the sort function in lisp (destructive)

single CONS cell, but imply passing an entire list or alist or tree etc. This implication of what we really intend to pass is *not* built into the language, but is implied by what function we are calling. For example, when calling the SORT or SORT-KEEPING-HEAD-CELL function we're really passing only the pointer to the head cell, but we imply passing the entire list. Kent Pittman's essay on intention of data type <<http://www.nhplace.com/kent/PS/EQUAL.html>> may be useful reading to grok this idea nearly in fullness.

If anyone ever encourages me I might write a companion essay that extends the idea into ambiguous atomic data types such as integers (used as if characters in emacs-lisp and c and most other languages except Common Lisp), and integers again (used as if bit vectors even in Common Lisp), and numbers in general (used as-is when really a physical unit is implied, such as kilometers or seconds or volts or ohms etc.). Hans Moravec had a nice MacSyma utility that used data values that carried the physical unit around with them, and thereby allowed trivial conversion from any unit to any other unit, his favorite being furlongs per fortnight as a unit for speed or velocity. (I personally prefer nano-c for walking/bicycling speeds, micro-c for aircraft speeds, etc.)

.