# 13. Locatives

## 13.1 Cells and Locatives

A *locative* is a type of Lisp object used as a *pointer* to a *cell*. Locatives are inherently a more "low level" construct than most Lisp objects; they require some knowledge of the nature of the Lisp implementation.

A *cell* is a machine word that can hold a (pointer to a) Lisp object. For example, a symbol has five cells: the print name cell, the value cell, the function cell, the property list cell, and the package cell. The value cell holds (a pointer to) the binding of the symbol, and so on. Also, an array leader of length *n* has *n* cells, and an **art-q** array of *n* elements has *n* cells. (Numeric arrays do not have cells in this sense.) A locative is an object that points to a cell; it lets you refer to a cell so that you can examine or alter its contents.

**contents** *locative*

Returns the contents of the cell which the locative points to. This is actually the same as **cdr**, for reasons explained below.

To modify the contents of the cell, use **setf** on **contents**:

```
(setf (contents loc) newvalue)
```

The macro **locf** (see page 271) can be used to convert a form that accesses a cell to one that creates a locative pointer to that cell: for example,

```
(locf (fsymeval x))
```

evaluates to a locative that points to the function cell of the value of **x**; that is to say, it points to the place where **(fsymeval x)** is stored.

**locf** is very convenient because it saves the writer and reader of a program from having to remember the names of many functions that would create locatives to cells found in different fashions.

One thing you should know is that it is not possible to make a locative to an element of a numeric array. For example,

```
(setq foo (make-array 10 ':type art-1b))
(locf (aref foo 0))
```

will get an error. Locatives may only point at entire words of memory, which contain standard Lisp data.

Because of cdr-coding (see section 5.4, page 72), a cons does not always contain an explicit cell which points to its cdr. Therefore, it is impossible to obtain a locative which points to such a cell. However, this is such a useful thing to do that *the cons itself* is usually treated as if it were a locative pointing to a cell which holds the cons's cdr. **(locf (cdr x))** returns the value of **x**, and **(contents x)** returns the cdr when *x* is a cons, so **(contents (locf (cdr x)))** is the same as **(cdr x)**, as it should be. Most functions that are normally given locatives will accept a cons as a "locative" to its cdr.

A cons always does contain a cell which points to the car, and (locf (car x)) will return a locative whose pointer field is the same as that of x's value.

## 13.2 Functions That Operate on Locatives

**location-boundp** *locative*

This returns t if the cell to which *locative* points contains anything except an "unbound" marker.

The unbound marker is a special data type, dtp-null, which is stored in cells to say that their value is missing. For example, an unbound variable actually has an unbound marker in its value cell, and (location-boundp (locf x)) is equivalent to (variable-boundp x).

**location-makunbound** *locative*

This stores an "unbound" marker into the cell to which *locative* points.

Other functions with which locatives are expected or useful include get (the locative points to the cell in which the plist is stored), store-conditional (the locative points to the cell to be tested and modified), and bind (the locative points to the cell to be bound).

## 13.3 Mixing Locatives with Lists

Either of the functions car and cdr (see page 61) may be given a locative, and will return the contents of the cell at which the locative points. They are both equivalent to contents when the argument is a locative.

Similarly, either of the functions rplaca and rplacd may be used to store an object into the cell at which a locative points.
For example,
```
    (rplaca locative y)
```
is the same as
```
    (setf (contents locative) y)
```

If you are just using locatives, you should use contents rather than car or cdr. But you can also mix locatives and conses. For example, the same variable may usefully sometimes have a locative as its value and sometimes a cons. Then it is useful that car and cdr work on locatives, and it also matters which one you use. Pick the one that is right for the case of a cons.

For example, the following function conses up a list in the forward order by adding onto the end. It needs to know where to put the pointer to the next cell. Usually it goes in the previous cell's cdr, but the first cell gets put in the cell where the list is supposed to end up. A locative is used as the pointer to this cell. The first time through the loop, the rplacd is equivalent to (setq res ...); on later times through the loop the rplacd tacks an additional cons onto the end of the list.

```
(defun simplified-version-of-mapcar (fcn lst)
  (do ((lst lst (cdr lst))
       (res nil)
       (loc (locf res)))
      ((null lst) res)
    (rplacd loc
            (setq loc (ncons (funcall fcn (car lst)))))))
```