Lisp Macros

Kevin McAllister

September 23, 2015

... a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.

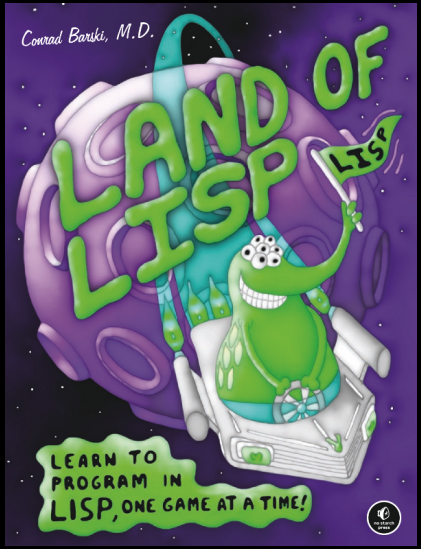
- Structure and Interpretation of Computer Programs

```
(defun double (x)
  (* x 2))
```

> *Common Lisp is a general-purpose, multi-paradigm programming language. It supports a combination of procedural, functional, and object-oriented programming paradigms.*

- Common Lisp (wikipedia)

- I'm just working through the examples in Land Of Lisp by Conrad Barski
- Chapter 16



- "In Lisp, programs are data, but the implications take a while to sink in."

– On Lisp by Paul Graham

```lisp
(defun add (a b)
  (let ((x (+ a b)))
    (format t "The sum is ~a" x)
    x))
```

- Contention the let and all it's parens is ugly

```lisp
(defmacro let1 (var val &body body)
  `(let ((,var ,val))
     ,@body))
```

```lisp
(defun add (a b)
  (let1 (x (+ a b))
    (format t "The sum is ~a" x)
    x))
```

Let's change this

```
(defun my-length (lst)
  (labels ((f (lst acc)
    (if lst
      (f (cdr lst) (1+ acc))
      acc)))
    (f lst 0)))
```

into

```
(defun my-length (lst)
  (recurse (lst lst acc 0)
    (split lst
      (self tail (1+ acc))
      acc)))
```

- Give me a split that can do a thing

```
(defmacro split (val yes no)
  (let1 g (gensym)
  `(let1 ,g ,val
    (if ,g
      (let ((head (car ,g))
            (tail (cdr ,g)))
        ,yes)
        ,no))))
```

- Hygenic and Anaphoric
  - g won't collide
  - head and tail will be available inside the split.

```
(defun pairs (lst)
  (labels ((f (lst acc)
             (split lst
               (if tail
                 (f (cdr tail)
                   (cons (cons head (car tail)) acc))
                 (reverse acc))
               (reverse acc))))
    (f lst nil)))

(pairs '(a b c d e f))
((A . B) (C . D) (E . F))
```

```
(defmacro recurse (vars &body body)
  (let1 p (pairs vars)
    `(labels ((self ,(mapcar #'car p) ,@body))
     (self ,@(mapcar #'cdr p)))))
```