

More Macros

More Macro Pitfalls

- Number of evaluation.
- Order of evaluation.
- Macro expanders with side effects.

Number of evaluations

- (defmacro for ((var start stop) &body body)
 `(do ((,var ,start (1+ ,var))
 ((> ,var ,stop))
 ,@body))
- (let ((x 2))
 (for (i 1 (incf x))
 (princ i)))

Order of evaluations

- (defmacro for ((var start stop) &body body)
 (let ((gstop (gensym)))
 `(do ((,gstop ,stop)
 (,var ,start (1+ ,var)))
 ((> ,var ,gstop))
 ,@body)))
- (let ((x 1))
 (for (i x (setq x 13)) (princ i)))

Expanders with effects

- (defmacro nil! (x)
 (incf *nil!s*)
 `(setf ,x nil))
- (defmacro our-trace (arg)
 (print arg)
 arg)

Expanders with effects

- ```
(defun rotate-char (char)
 (unless (alpha-char-p char)
 (return-from rotate-char char))
 (let ((code (char-code char))
 (base (if (upper-case-p char)
 (char-code #\A)
 (char-code #\a))))
 (code-char
 (+ base (mod (+ 13 (- code base)) 26)))))
```

# Expanders with effects

---

- ```
(defmacro rot13 (string)
  (dotimes (i (length string) string)
    (setf (aref string i)
          (rotate-char (aref string i))))))
```
- ```
(rot13 "This is a test!")
```

# Uses for Macros

---

- Implicit quoting.
- Cosmetics.
- Controlling evaluation.
- Syntactic abstraction.
- Side effects.
- Efficiency.

# Implicit quoting.

---

- `(defun f (x) (+ x x))`
- `(setf (fdefinition 'f)  
 (lambda (x) (+ x x)))`

# Cosmetics

---

- `(let ((x 42) (y 4711))  
 (+ x y))`
- `((lambda (x y) (+ x y)) 42 4711)`

# Evaluation Control

---

- Conditional evaluation: if, cond, when, unless, when-let, etc.
- Delayed evaluation: delay, force, run-in-thread, etc.

# Syntactic Abstraction

---

- Hiding implementation details.

# Side effects

---

- Functions don't take reference parameters.
- So only macros can modify variables that are passed as arguments.

# Efficiency

---

- ```
(defmacro my-add (arg1 arg2)
  (if (and (numberp arg1) (numberp arg2))
      (+ arg1 arg2)
      `(+ ,arg1 ,arg2)))
```
- [Better do this with compiler macros!]