COMP 210, FALL 2000
Lecture 9: Even More Tricks with Lists

Reminders:
- Homework assignment 1 available today, due Wednesday, February 9, 2000
- Exam will be 2/16/2000, in class–closed-notes, closed-book

Review
1. Did more work with lists, introduced Scheme's built-in list construct, based on **cons**, **first**, **rest**, and **cons?**
2. Talked some about data definitions and when we need them. At this point in COMP 210, we want you to write a data definition for each list construct (even though you use **cons** *et al.*) because the data definition specifies what kind of object is going into the list (list-of-symbol versus list-of-natnum versus list-of-plane).
3. Talked some about templates. The template relates directly to a data-definition. We write a separate template for each kind of information that needs a data definition.

Take Home Problem
Recall that we defined a whole bunch of data for brands and planes for JetSet Airlines.

```
;; a brand is a
;;   (make-brand type speed seats service)
;; where type is a a symbol, and speed, seats, and service
;;       are numbers
(define-struct brand (type speed seats service))

;; a plane is a
;;    (make-plane  tailnum a-brand miles mechanic)
;; where tailnum is a symbol, a-brand is a brand
;;       miles is a number, and mechanic is a list-of-symbols
(define-struct plane (tailnum a-brand miles mechanic))

;; a list-of-symbol is either
;;   - empty, or
;;   - (cons first rest)
;; where first is a symbol and rest is a list-of-symbol
;; [No define-struct since this uses cons, first, and rest]

;; a list-of-plane is either
;;   - empty, or
;;   - (cons first rest)
;; where first is a plane and rest is a list-of-plane
;; [No define-struct since this uses cons, first, and rest]
```

```
;; Example Data
;;
(define brand1 (make-brand 'DC-10   550  282   15000))
(define brand2 (make-brand 'MD-80  505  141   10000))
(define brand3 (make-brand 'ATR-72 300   46    5000))

;;
(define N1701 (make-plane 'N1701 brand1 0 empty))
(define N3217 (make-plane 'N3217 brand2 100
                    (cons 'Susan (cons 'Mike (cons 'Pam empty)))
                    ))
(define N1079 (make-plane 'N1079 brand1 3500
                    (cons 'Mike (cons 'Bubba (cons 'Susan empty)))
                    ))
(define N9824 (make-plane 'N9824 brand3 500
                    (cons 'Bess (cons 'Felix (cons 'Jane empty)))
                    ))
(define N3141 (make-plane 'N3141 brand3 1000
                    (cons 'Fred (cons 'Bubba empty))
                    ))
;;
(define JetSetFleet
  (cons N1701 (cons N3217 (cons N1079 (cons N9824 (cons N3141 empty)))))
  )
```

At the end of class, I asked you to solve the following problem. Write a program that consumes a list-of-planes and produces a list containing all the planes that are DC-10s. This is my version, one student produced an equivalent version for class.

```
;; just-DC-10s: list-of-plane -> number
;; Purpose:  consumes a list-of-plane and returns the number that
;;          are DC-10s
(define (count-DC-10s a-lop)
  (cond
    [(empty? a-lop)  0]
    [(cons?  a-lop)
       (cond
         [(symbol=? (brand-type (plane-a-brand(first a-lop))) 'DC-10)
          (add1 (count-DC-10s (rest a-lop)))]
         [else  (count-DC-10s (rest a-lop))])
       ]
  ))
```

An alternative way to write this program, suggested to me by one of the students, is

```
;; just-DC-10s: list-of-plane -> number
;; Purpose:  consumes a list-of-plane and returns the number that
;;          are DC-10s
(define (count-DC-10s a-lop)
  (cond
    [(empty? a-lop)  0]
    [(cons?  a-lop)
      (add
          (cond
             [(symbol=? (brand-type (plane-a-brand(first a-lop))) 'DC-10)  1]
             [else 0] )
          (count-DC-10s (rest a-lop))] ) ]
  ))
```

Is this acceptable? This brings us back to the heart of COMP 210. COMP 210 is a course about a bottom-up, data-driven, design methodology for programming in the small. This alternative formulation will produce the correct results, but it is not the code that the methodology would generate. It is clever, but it is not the code that the methodology would generate. Thus, in COMP 210, it is **not** the code that you should write.

[Yes, it contains one fewer textual copy of "(count-DC-10s (rest a-lop))." That does not make it run faster. That does not make it inherently better. In our judgement, the former version is easier to understand, easier to go back and read ten years later, and (probably) easier to modify. Equally important, from the COMP 210 perspective, it is the code that the design methodology will generate, and COMP 210 is a course about the design methodology.]

A More Complex Variation

Write a program **all-the-brand** that consumes a list-of-plane and a symbol and produces a list-of-plane containing all the planes whose brand matches the symbol. Build on your knowledge from **just-dc10s**. You can use the same data-definitions and example data.

```
;; all-the-brand : list-of-plane symbol -> list-of-plane
;; Purpose: consumes a list-of-plane and produces a list-of-plane
;;         that contains all the planes whose type matches the
;;         second argument
(define (all-the-brand a-lop kind) … )
```

```
;; Templates
;;
;; for brand
;; (define ( ... a-brand ... )
;;   ( ... (brand-type    a-brand) ...
;;     ... (brand-speed   a-brand) ...
;;     ... (brand-seats   a-brand) ...
;;     ... (brand-service a-brand) ... ))
;;
;; for plane
;; (define ( ... a-plane ... )
;;   ( ... (plane-tailnum   a-plane) ...
;;     ... (plane-a-brand   a-plane) ...
;;     ... (plane-miles     a-plane) ...
;;     ... (plane-mechanics a-plane) ... ))
;;
;; for list-of-symbols
;; (define ( ... a-los ... )
;;   (cond
;;     [(empty? a-los) ... ]
;;     [(cons?  a-los) ... (first a-los) ... (rest a-los) ... ]
;;   ))
;;
;; for list-of-planes
;; (define ( ... a-lop ...)
;;   (cond
;;     [(empty? a-lop) ... ]
;;     [(cons?  a-lop) ... (first a-lop) ... (rest a-lop) ... ]
;;   ))


;; all-the-brand : list-of-plane symbol -> list-of-plane
;; Purpose: consumes a list-of-plane and produces a list-of-plane
;;          that contains all the planes whose type matches the
;;          second argument
(define (all-the-brand a-lop kind)
  (cond
    [(empty? a-lop)    empty]
    [(cons?  a-lop)
       (cond
         [(symbol=? (brand-type (plane-a-brand (first a-lop))) kind)
          (cons (first a-lop) (all-the-brand (rest a-lop) kind)) ]
         [else
          (all-the-brand (rest a-lop) kind)]
       )]
    ))
```