Extra topics, Week 11

Overview

- Prelim.
  - Admin.
  - About the quiz.
- Q & A
  - Vague question about association lists
  - How do we count procedure calls?

Admin

- Upcoming extra credit
  - See yesterday's class

What might be on the quiz?

- Association Lists
- Higher order procedures (?)
- Analyzing procedures

Association Lists

- Useful to association a bunch of information with a keyword
  - Your student entry with your ID number
  - Information about a course based on its name
  - Your purchase history based on your email address
- Two typical tasks:
  - Represent the info
  - Write procedures that, using that representation, let us quickly access part of the info
- Scheme has a quick and easy to understand solution (that is slow)
- Representation: Association List: List of key/info pairs
  - (list (list 123456 "Nazhay" "Ehtinak" "A" "A" "C" "A") (list 521356 "Nancy" "Ehtinak" "B" "B" "A" "B")
  - (list (list "CSC151" "Time-consuming" "Strange" "Invaluable") (list "TUT101" "Essential" "Highly variable") (list "ECN495" "Wealth generating"))
  - (list (list "cbw@grinnell.edu" "Strange t-shirt" "Macbook" "Vicodin"))
- Quickly find things by key: assoc. (assoc key list) -> Find the first entry that starts with the key

Implementation is really straightforward

```
(define assoc
  (lambda (key alst)
    (cond
      [(equal? key (caar alst))
       (car alst)]
      [else (assoc key (cdr alst))])))
```

Whoops? What if the list is empty? (That is, what if the element doesn't appear?) Early design decision: Return false.

```
(define assoc
  (lambda (key alst)
    (cond
      [(null? alst)
       #f]
      [(equal? key (caar alst))
       (car alst)]
      [else (assoc key (cdr alst))])))
```

How do we use it?

- Almost exclusively to look up values in a table
- Scheme interpreters - What value is associated with this name?
- But it's limited.
- Our questions are generally more general. Rather than "Tell me about CSC151", it's "Tell me about the writing-intensive classes."

Sample data

```
(define some-grinnell-courses
  (list
    (list "CSC151" "Hard" "Time-consuming" "Artistic" "Crash" "Mathy")
    (list "TUT101" "Variable" "Necessary" "Writing-intensive")
    (list "SPN105" "Time-consuming" "Talkative" "Memory-intensive")
    (list "ANT1xx" "Mind-expanding" "Scientific")
    (list "MUS1xx" "Interesting" "EZA")
    (list "UM100" "Fictitious" "EZA" "Writing-intensive")))

(define lookup-courses-that-meet-paritular-criteria
  (lambda (key alst)
    (lookup-courses key alst)))

;;; Find a list of all courses that meet a particular criterion
;;; (that is, that include the word in the description list)
(define lookup-courses
  (lambda (criterion courses)
    (cond
      [(null? courses)
       null]
      [(member? criterion (cdar courses))
       (cons (car courses) (lookup-courses criterion (cdr courses))]
      [else (lookup-course criterion (cdr courses))])))
```

```
(define member?
  (lambda (val lst)
    (and (not (null? lst))
         (or (equal? val (car lst))
             (member? val (cdr lst))))))
```

Counting Procedure Calls

- Sometimes we do this by hand. "How many times does lookup-courses get called for a list of 10 courses?"
  - It has to look at each element. That’s 10 calls.
  - It has to look at the empty list. THat’s another.
  - So 11 calls.
- What if it’s not quite so obvious? Make the computer count!
- Steps
  - First, create a counter (define counter (vector-new 0))
  - Provide a way to increment the vector (define increment (lambda () (vector-set! counter 0 (+ 1 (vector-ref counter 0))))
    - Get
    - Increment
    - Store
  - Annotate/expand your procedures to cont (define myproc (lambda (params) (increment) body))

Six P’s for the Project

```
;;; Procedure:
;;;   image-series
;;; Parameters:
;;;   n, an integer
;;;   width, a positive integer
;;;   height, a positive integer
;;; Purpose:
;;;   Create an "interesting" image.
;;; Produces:
;;;   image, an image
;;; Preconditions:
;;;   0 <= n < 1000
;;; Postconditions:
;;;   (image-width image) = width
;;;   (image-height image) = height
;;;   For all m != n
;;;     (image-series m width height) is different than image
```

Yes, you should document each procedure you call. (Six P’s are better, but even just a short purpose would be okay.)

```
;;; Make a "fence" that has the given number of slats
(define make-fence
  (lambda (image number-of-slats)
    ....))

;;; Cut a hole in the fence at position (x,y)
(define holey-fence
  (lambda (image x y)
     ...))
```

Folks who showed up today (YK, HF, WBC, ZC) and members of their groups do not need to make sure that turtle operations scale appropriately if the aspect ratio of the image changes. (However, they should scale appropriately if the aspect ratio stays the same and the image changes.)

Samuel A. Rebelsky, rebelsky@grinnell.edu