Week 2 Lecture 5

Theory

Getting Ready

- Feel good about Lecture 4
- Read SICP Section 2.2 closely

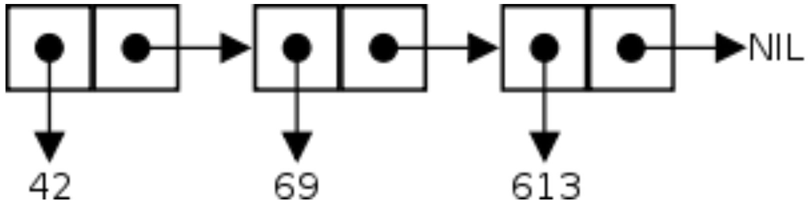
What's in this lecture?

- Lists and Recursion over Data Structures in Scheme

Lists

- Lists are a *sequential* data structure, similar to an Array
- Unlike an Array, list elements are arranged in a *chain* of cons cells (pairs)
- Each cons cell list element has a *value* and a *next* pointer

Lists



- This is the list (42, 69, 613)
- NIL terminates the end of the list
- The empty list itself is just NIL (which can also be written as ())

Making a List

```
(cons 1 (cons 2 ()))

(cons “hello” (cons “world” ()))


(define list1 (cons 1 (cons 2 ()))
```

Lists

```
()

‘(1 2 3)

‘(“hello” “world”)

‘(“foo” 2 “baz”)
```

Processing a List

- Previously we used numeric tests as the recursion base case (such as a > b)
- Now, we use *structure* as a base case; terminate when the list is empty
- The general formula is: process the head of the list (car theList), and recurse on the tail of the list (cdr theList)

Length of a List

```
(define (length alist)
  (define (list-iter alist count)
    (if (= alist ())
      count
      (list-iter (cdr alist) (+ 1 count))))
  (list-iter alist 0))
```

Concatenate Lists

How do we join two lists together?

```
(def (concat a b)
  (if (= a ())
    b
    (cons (car a) (concat (cdr a) b))))
```

Now you try...

How would you implement *contains*, which returns #t if the list contains the element?

(define (contains a elem) ...)

How would you implement *reverse* of a list?

(define (reverse a) ...)

Exercises

- Read SICP 2.2.1 closely
- SICP 2.17, 2.21, 2.23