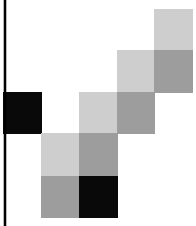


Data Abstraction…

The truth comes out…

What we're doing today…

- Abstraction
- ADT: Dotted Pair
- ADT: List
- Box and Pointer
- List Recursion
- Deep List Recursion

Administrivia!

- Midterm 1 will be graded by Saturday. Expect seeing a grade some time this weekend.
- Project 1 grades should be coming in soon. Just hold onto your horses =)

Abstraction

- The BIGGEST idea of this course
- Ability to hide the lower levels of detail
- Example:
  - Driving a car but not knowing how it really runs
  - Using sentences and words, but not knowing exactly how it's implemented….until now.

Abstract Data Type (ADT) is…

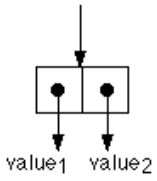
- the logical data structure itself (an abstraction, not the detailed implementation), combined with…
- a set of operations which work on the data structure.
- When we use ADTs, we don't care how they're implemented, just how to use them.

ADT: The Dotted Pair

- What is a Pair?
  - Most basic data structure
  - Puts two things together

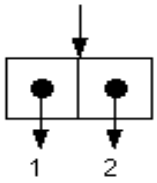
ADT: The Dotted Pair

- Constructor:
  - Cons: (cons <*value*$_1$> <*value*$_2$>)
    - Example:
      - (cons 1 2) => (1 . 2)



ADT: The Dotted Pair

- Selectors:
  - Car: (car <*cons-cell*>)
    - Examples:
      - (car (cons 1 2)) => 1
  - Cdr: (cdr <*cons-cell*>)
    - Examples:
      - (cdr (cons 1 2)) => 2



ADT: Lists 1/7

- What are Lists?
  - An ordered set of elements enclosed in '( )'
  - Built on cons cells, so it's a pair whose 'cdr' is the empty list
    - (list <value$_1$> ... <value$_n$>) =>
      (cons <value$_1$> … (cons <value$_n$> nil))

ADT: List 2/7

- Difference between lists and sentences?
  - A sentence can contain only words and sentences
  - A list can contain ***anything***.
    - Booleans
    - Procedures
    - Other lists
- Sentences can be thought of as a "flat" list.
- They both have their own set of constructors and selectors.

ADT: Lists 3/7

- More Constructors & Examples:
  - Cons
    - Examples:
      - (cons 'a 'b) => (a . b)
      - (cons (cons a '()) (cons b (cons c '()))) => ((a) b c)
  - List
    - Examples:
      - (list 'a 'b) => (a b)
      - (list (cons a (list b) (list c)) => ((a b) (c))
  - Append: ***always*** takes in lists.
    - Examples:
      - (append (list a b) (list c d)) => (a b c d)
      - (append (list (list a b)) (list c d)) => ((a b) c d)

ADT: Lists 4/7

- Higher Order Functions
  - Map (like *every*)
    - Usage: (map *<unary function> <list>*)
    - Example:
      - (map (lambda (x) (list x)) '(1 2 3 4))
        => ((1) (2) (3) (4))
  - Filter (like *keep*)
    - Usage (filter *<pred?> <list>*)
      - (filter list? '(a (b c) () ((d)))
        => ((b c) () ((d)))
  - Reduce (like *accumulate*)
    - Usage (reduce *<binary function> <list>*)
      - (reduce (lambda (x y) (if (> (length x) (length y)) x y)) '(a (b c d) () ((e)))
        => (b c d)

ADT: Lists 5/7

- More Primitives for Lists!
  - **length**: returns the number of elements in a list (like *count*)
    - Usage: (length *<list>*)
  - **null?**: returns #t if it's an empty list otherwise #f (like *empty?*)
    - Usage: (null? *<list>*)
  - **list?**: returns #t if argument is a list, #f otherwise
    - Usage: (list? *<list>*)

ADT: Lists 6/7

- **list-ref**: returns the element at that position where the first element is the 0th position. (like *item*)
  - Usage: (list-ref *<position>* *<list>*)
- **equal?**: works the same way as with sentences.
  - Usage: (equal? *<list>*)
- **member**: returns the part of the list starting with the element, otherwise #f
  - Usage: (member *<element>* *<list>*)

ADT: Lists 7/7

- Use them correctly!
  - Examples:
    - (se (list 'this 'is 'bad) (list 1 2 3))
    - (first (list 'this 'is 'a 'list))
    - (car (se 'this 'is '(a sentence)))
- Yes they may produce the correct results but it's a…

DAV!

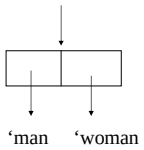
- Data Abstraction Violation
- You'll probably only hear this in 61a, but…
- We will DING you guys on this on…
  - Homework
  - Projects, and especially
  - Exams!

ADTs

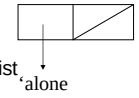
- Whenever creating a new ADT make sure:
  - You have constructors to create that data type
  - You have selectors for that data type
- Example:
  - Create a new data type car which takes a driver, navigator, and a passenger
  - Create appropriate constructors and selectors for this data type.
  - Create a procedure run-around which returns a new car with the driver as the old car's navigator, the navigator as the passenger, and the passenger as the driver.

Box and Pointers 1/5

- Back to Pairs
  - (cons 'man 'woman) ;; a pair
    - cons simply makes pairs



- One Element List
  - (cons 'alone '()) ;; a list
    - cons to null at the end makes list

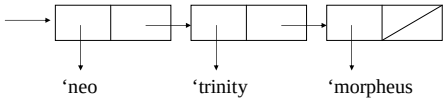


Box and Pointers 2/5

- Tips on how to do box and pointer diagrams.
  - See how many elements are in the list. This is the "backbone" of the box and pointer diagram.
  - Whenever you see cons…draw a box.
  - If you can write out the whole 'cons' representation of the list. This will show you how many boxes you'll be drawing.
  - Draw a starting arrow to the first box.
  - Just take it by 'car' and 'cdr' and you'll be fine! ☺
  - Whenever you see a dotted pair, it means that it's not a list, so it shouldn't end with a null box.

Box and Pointers 3/5

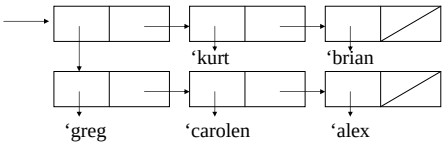
- A Longer List
  - (cons 'neo (cons 'trinity (cons 'morpheus '())))
    - Equivalent to (list 'neo 'trinity 'morpheus)



- Whenever you see the list procedure, go ahead and build a 'backbone'

Box and Pointers 4/5

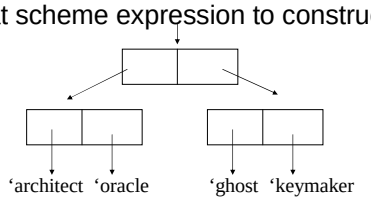
- A Nested List
  - (cons (list ‘greg ‘carolen ‘alex) (list ‘kurt ‘brian))



- ‘((greg carolen alex) kurt brian)

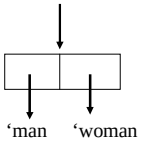
Box and Pointers 5/5

- Just to make sure you didn’t forget about pairs…
- What scheme expression to construct...



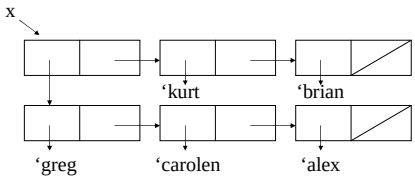
Chasing cars… and cdrs? 1/3

- car means follow the first arrow
- cdr means follow the second arrow



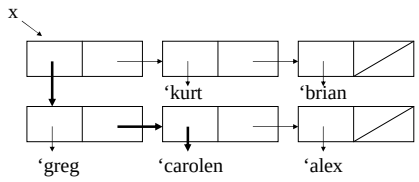
Chasing cars… and cdrs? 2/3

- (define x ‘((greg carolen alex) kurt brian))
- (car (cdr (car x))) ;; what is this?



Chasing cars… and cdrs? 3/3

- (define x '((greg carolen alex) kurt brian))
- (car (cdr (car x))) ;; what is this?



List, List, List…(List recursion!)

- List recursion is the same as sentence recursion, but using the list constructors and selectors.
  - Example:

```
(define (foo lst)
    (if (null? lst)
        nil
        (cons (car lst)
              (foo (cdr lst)))))
(define (foo2 lst)
    (if (null? lst)
       nil
       (append (list (car lst))
               (foo (cdr lst)))))
```

- Reversing a list is a little bit different than with sentences. This was in lab.
- Once you get enough practice with list recursion, it'll become second nature.

Deep List Recursion 1/3

- This is really simple.
- Just do normal recursion, and stick in another base case that asks if the argument is a list.
- Example:
  -

```
(define (square-list L)
    (if (null? L)
       nil
       (cons (square (car L)) (square-list (cdr L)))))
```

  -

```
(define (square-deep-list L)
    (cond ((null? L) nil)
          ((list? (car L)) (cons (square-deep-list (car L))
                                 (square-deep-list (cdr L))))
          (else (cons (square (car L))
                      (square-deep-list (cdr L))))))
```

Deep Recursion 2/3

- Write a function 'rev' that reverses a list
  - (rev '(1 2 (3 4))) => ((3 4) 2 1)
- Now make it so that it does a deep reverse
  - (deep-rev '(1 2 (3 4))) => ((4 3) 2 1)

Deep Recursion 3/3

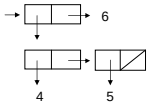
- Answer
  - (define (rev l)
    (if (null? l)
    '()
    (append (rev (cdr l)) (list (car l)))))
  - (define (deep-rev l)
    (cond ((null? l) '())
    ((list? (car l)) (append (deep-rev (cdr l))
    (list (deep-rev (car l)))))
    (else (append (deep-rev (cdr l))
    (list (car l))))))

More Problems…

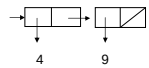
- Box and Pointer Practice! (Write what each evaluates to and the box and pointer for each expression)
  - (cons (list 4 5) 6)
  - (append (cons 4 '( )) (list 9))
  - (list 4 (list 5) (list 6))
- Write the cons representation of this list and the box and pointer:
  - (2 ((3) 4) ((5 . 6)) 7 . 8)

More Practice Answers! 1/2

- (cons (list 4 5) 6) → ((4 5) . 6)

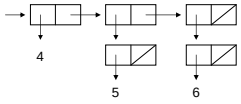


- (append (cons 4 '( )) (list 9)) → (4 9)

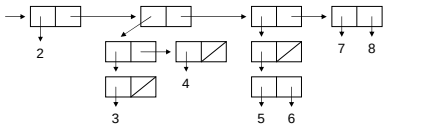


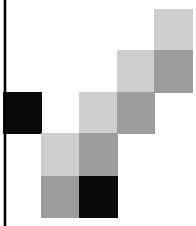
More Practice Answers 2/2

- (list 4 (list 5) (list 6)) → (4 (5) (6))



- (2 ((3) 4) ((5 . 6)) 7 . 8)
  → (cons 2 (cons (cons (cons 3 nil) (cons 4 nil))
  (cons (cons (cons 5 6) nil) (cons 7 8))))





Have a Great Weekend! Relax you've earned it!

...full of scheme fun ☺