Recap

Expressions
- (+ 1 2 3) → 6

Lists
- nil
- (cons 'A '(B C)) → (A B C)
- (car '(A B C)) → A
- (cdr '(A B C)) → (B C)
- (car (cdr '(A B C))) → B
- also have length, append, reverse

Recap (2)

Conditional
- (cond (test_1 result_1) ... (test_n result_n))
- tests:
  * (eq x y)
  * (< x y)
  * (atom x) (will want to use in problem set)

Functions
- (defun cube (x) (* x x x))
- [sheep]
- (division)
- (wrap up insertion-sort)

Notes

- Book uses slightly different dialect of Lisp
- Use class notes as a reference, or the Lisp tutorial on the web site.

Major differences
- use T instead of true
- use (defun ...) instead of (define ...)
- use (mapcar #\' f l) instead of (maplist f l)
**Insert**

- Insert a number into a sorted list

- \( \text{insert } 4 \ '(1 \ 3 \ 7) \rightarrow (1 \ 3 \ 4 \ 7) \)

- \( \text{defun insert } (x \ l) \)
  \( \quad (\text{cond } ((\text{eq } l \ \text{nil}) \ (\text{cons } x \ \text{nil})) \)
  \( \quad \quad (\langle x \ (\text{car } l)) \ (\text{cons } x \ l)) \)
  \( \quad \quad (t \ (\text{cons}) \)
  \( \quad \quad \quad \ (\text{car } l) \)
  \( \quad \quad \quad \ (\text{insert } x \ (\text{cdr } l)))) \))

**Encoding Trees**

- ' (Cow (Badger (Aardvark nil nil) nil)
  (Eel (Donkey nil nil) (Fox nil nil)))

**Encoding Records**

```java
class Book {
    String author;
    String title;
    int year;
    ...
}
```
Encoding Book Records

List Form: (Author Title Year)

ex: (McCarthy Lisp 1960)

(defun author (book) (car book))
(defun title (book)  (car (cdr book)))
(defun year (book)   (car (cdr (cdr book))))

(author '(McCarthy Lisp 1960)) -> McCarthy
(title '(McCarthy Lisp 1960)) -> Lisp
(year '(McCarthy Lisp 1960)) -> 1960

Encoding Records

books: '((Joyce Ulysses 1922)
(Rowling Harry-Potter 1997)
(McCarthy Lisp 1960)
...)

(titles books) -> (Ulysses Harry-Potter Lisp ..)
(years books) -> (1922 1997 1960 ..)

Lisp Memory Model

- Cons cell:
- Atom:
- (cons 'A (cons 'B (cons 'C nil)))

Sharing

(a) (b)

- Both structures could be printed as (A.B).(A.B)
- Which is result of evaluating (cons (cons 'A 'B) (cons 'A 'B))?
Programs As Data

(defun substitute (to from term)
  (cond ((atom term)
           (cond ((eq term from) to)
                  (t term)))
        (t (cons (substitute to from (car term))
                (substitute to from (cdr term))))))

(substitute 3 'w '(* (- 5 w) (+ w w)))
  is (+ (- 5 3) (* 3 3))

Programs As Data

(defun substitute-and-eval (to from term)
  (eval (substitute to from term)))

(substitute-and-eval '* '+ '(+ 10 2 3))
  evaluates to 60

(derivative '(* 3 x)) -> '(* 6 x)

(substitute-and-eval 6
  'x
  (derivative '(* 3 x))) -> 36
Genetic Programming

(defun move (...
  (cond (test ...)
        (t (m ...))))
  (f (+ ...))
  (g (h ...))))

(defun move (...
  (m (h (f (+ ...)))
       (cond ...))))

Mutate