Lisp (2)

CSCI 334
Stephen Freund

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)

- Condtional
  - (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

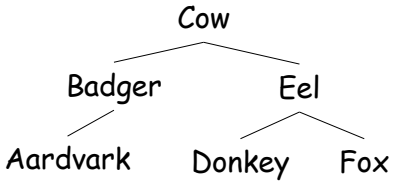
- Book uses slightly different dialect of Lisp
- Use class notes as a reference, or the Lisp tutorial on the web site.
- Major differences
  - use T instead of true
  - use (defun ...) instead of (define ...)
  - use (mapcar #'f l) instead of (maplist f l)

Insert

- Insert a number into a sorted list
- `(insert 4 '(1 3 7)) -> (1 3 4 7)`
-
```
(defun insert (x l)
   (cond ((eq l nil) (cons x nil))
         ((< x (car l)) (cons x l))
         (t (cons
              (car l)
              (insert x (cdr l))))))
```

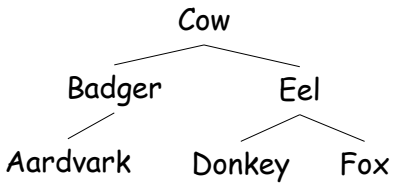
13

Encoding Trees



14

Encoding Trees




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

15

Encoding Records

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

16

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
```

17

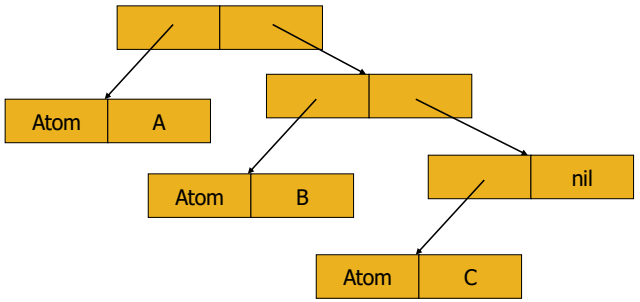
Encoding Records

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

(titles books)
  -> (Ulysses Harry-Potter Lisp ..)

(years books) -> (1922 1997 1960 ..)
```

18

Lisp Memory Model

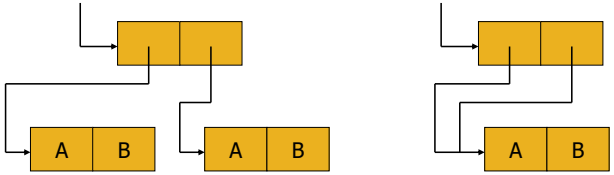
- Cons cell: Address | Decrement
- Atom: Atom | value
- (cons 'A (cons 'B (cons 'C nil)))



19

Sharing

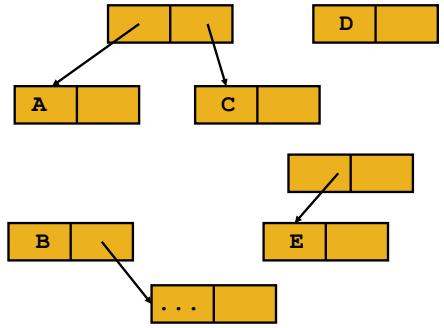
(a) (b)



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

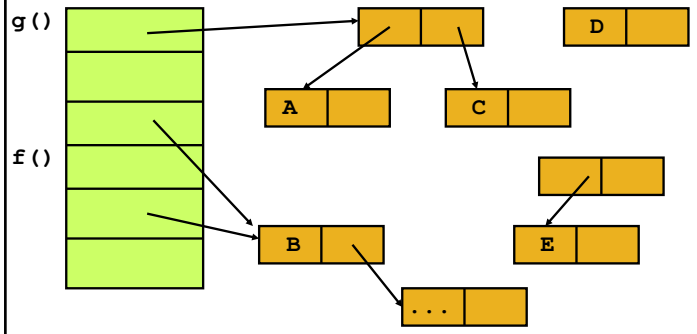
20

Garbage Collection



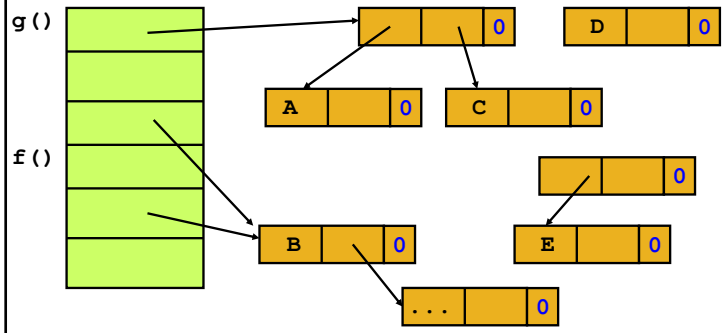
21

Garbage Collection



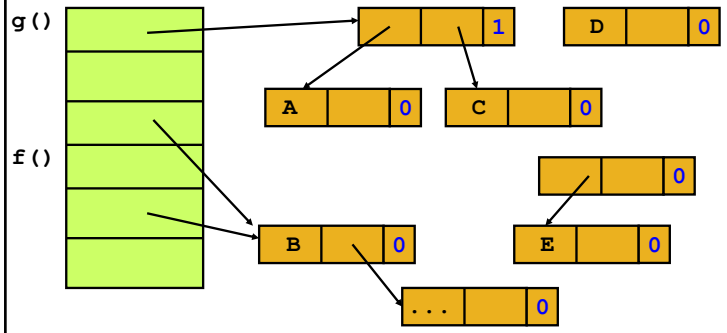
22

Clear Tags



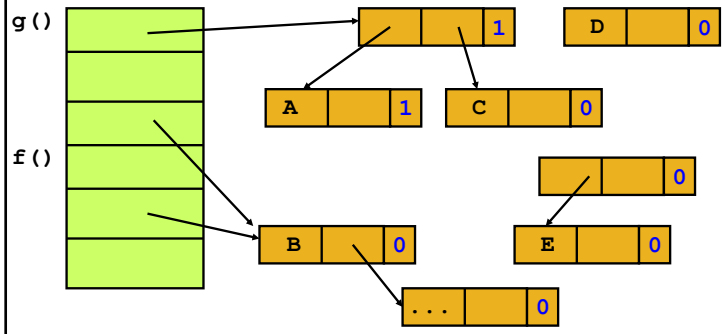
23

Mark Reachable Cells



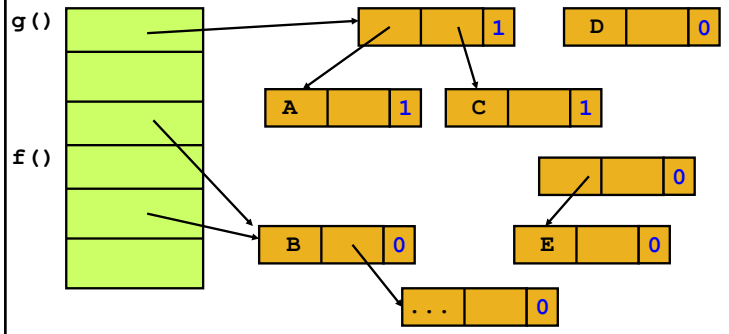
24

Mark Reachable Cells



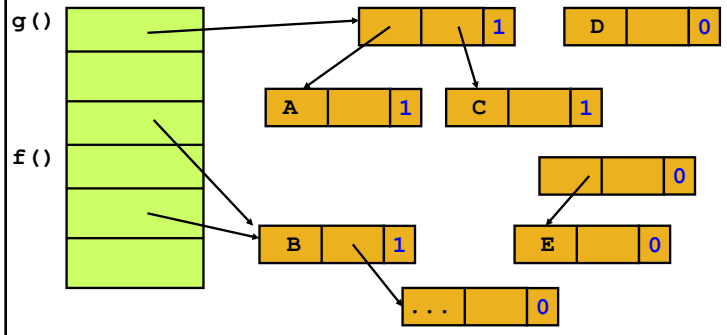
25

Mark Reachable Cells



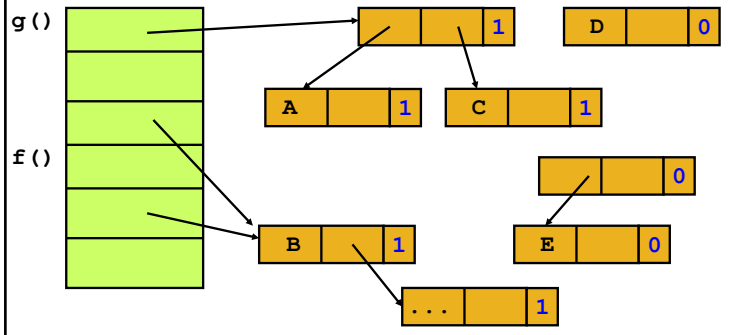
26

Mark Reachable Cells



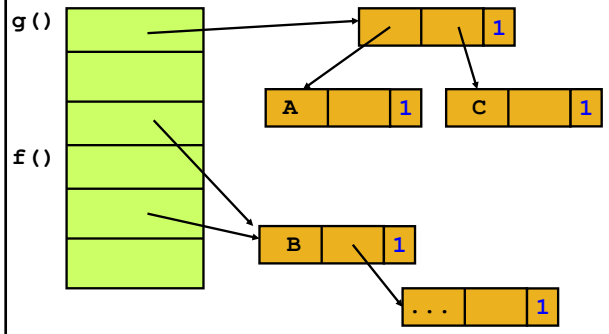
27

Mark Reachable Cells



28

Free Unreachable Cells



29

Programs As Data

```
; substitute "to" for "from" in "term"
(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))
```

30

Programs As Data

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

(substitute-and-eval '* '+ '(+ 10 2 3))
```

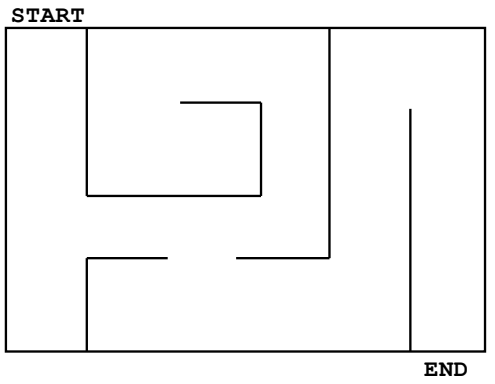
evaluates to 60

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

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

31

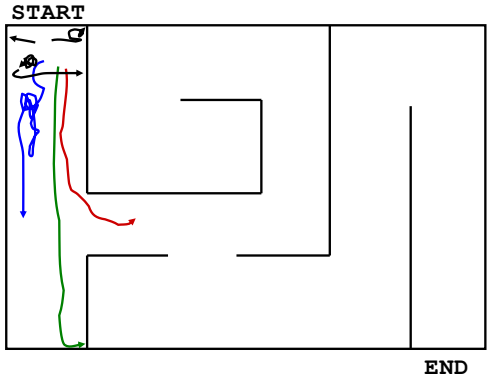
Genetic Programming

GP Example



32

Genetic Programming



33

Genetic Programming

```
(defun move (...)
  (cond (test ...)
        (t (m ...))))

(defun move (...)
  (f (+ ...)
     (...)))

(defun move (...)
  (g (h ...)
     (...))))
```

Mutate

```
...

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

(defun move (...)
  (f (test  ...)
     (h ...)))

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

...
```

34