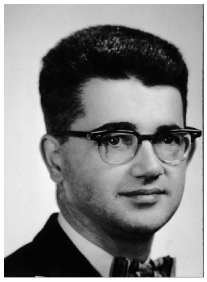
Lisp

CSCI 334
Stephen Freund

John McCarthy



IBM 704

Factorial

- Two cases:
  - base case: n is 0
  - recursive case: n * fact(n-1)

```
• (defun fact (n)
      (cond ((eq n 0) 1)
            (t (* n (fact (- n 1))))))
```

• `(fact 3) -> 6`

List Length

- Two cases
  - base case: empty list
  - recursive case: process "car", recurse on "cdr"

```
• (defun length (l)
      (cond ((eq l nil) 0)
            (t (+ 1 (length (cdr l))))))
```


Membership

- Does a list contain a specific atom?

• `(contains 'A '(C A M P)) -> t`
• `(contains 'B '(C A M P)) -> nil`

```
• (defun contains (x l)
      (cond ((eq x nil) nil)
            ((eq x (car l)) t)
            (t (contains x (cdr l)))))
```

Lisp (2)

CSCI 334
Stephen Freund

7

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

8

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)

9

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)

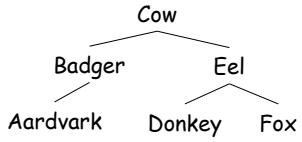
10

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))))))
  ```

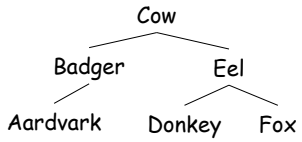
11

Encoding Trees



12
  ```

Encoding Trees




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


Encoding Records

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


Encoding Book Records

```
List Form: (Author Title Year)

      ex:  (Melville Moby-Dick 1851)

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

(author '(Melville Moby-Dick 1851)) -> Melville
(title '(Melville Moby-Dick 1851)) -> Moby-Dick
(year '(Melville Moby-Dick 1851)) -> 1851
```


Encoding Records

```
books: '((Joyce    Ulysses   1922)
         (Melville Moby-Dick 1851)
         (McCarthy Lisp      1960)
         ...)

(titles books) -> (Moby-Dick Ulysses Lisp ..)

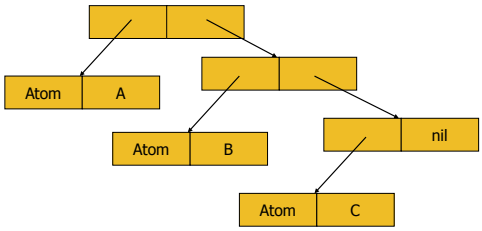
(years books) -> (1922 1851 1960 ..)
```


Lisp Memory Model

- Cons cell: 

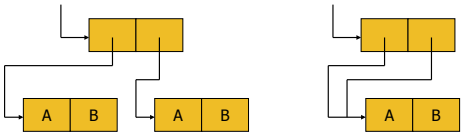
- Atom: 

- (cons 'A (cons 'B (cons 'C nil)))



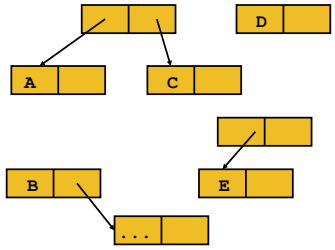
Sharing

(a) (b)



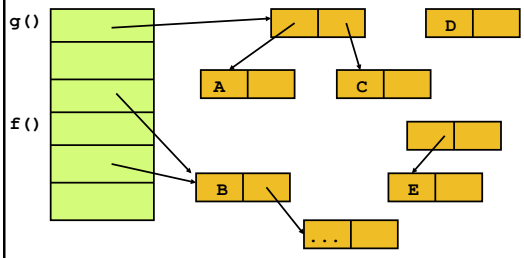
- Both structures could be printed as (A.B).(A.B)
- Which is result of evaluating
  (cons (cons 'A 'B) (cons 'A 'B)) ?

Clear Tags



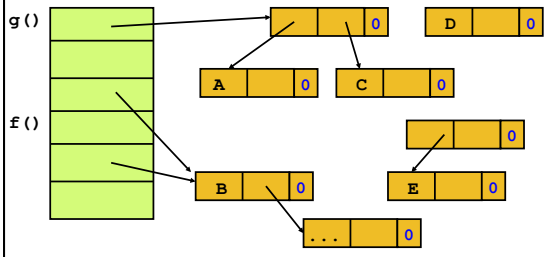
19

Clear Tags



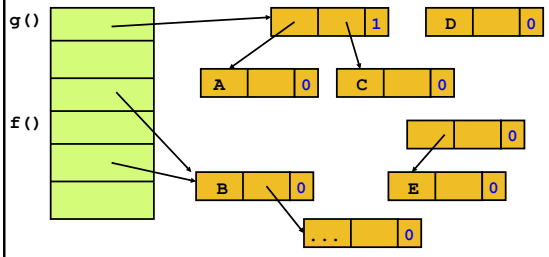
20

Clear Tags



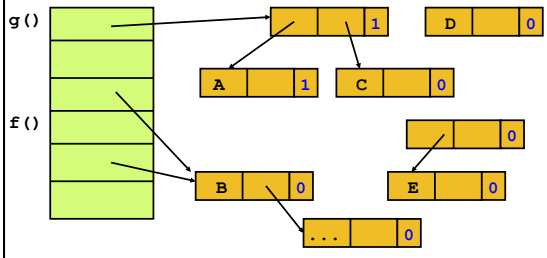
21

Mark Reachable Cells



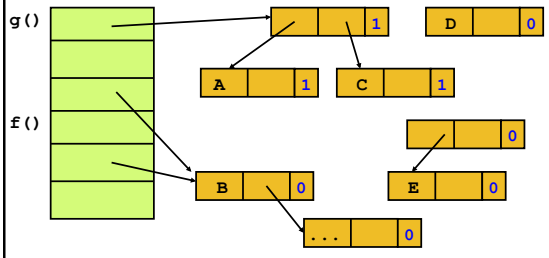
22

Mark Reachable Cells



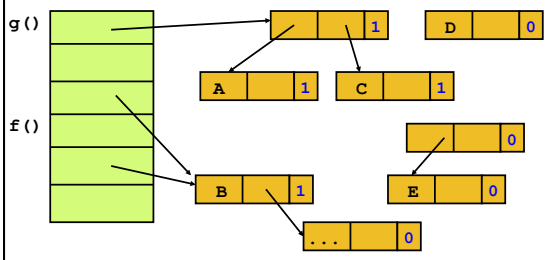
23

Mark Reachable Cells



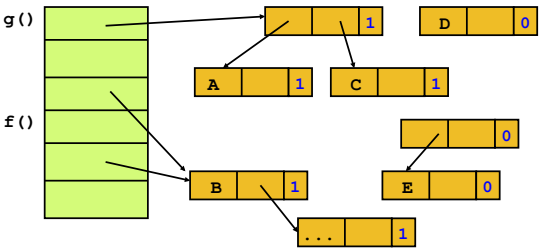
24

Mark Reachable Cells



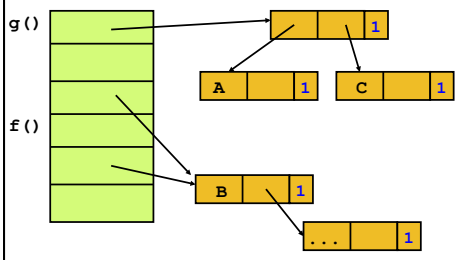
25

Mark Reachable Cells



26

Free Unreachable Cells



27

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))
```

28

Programs As Data

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

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

evaluates to 60

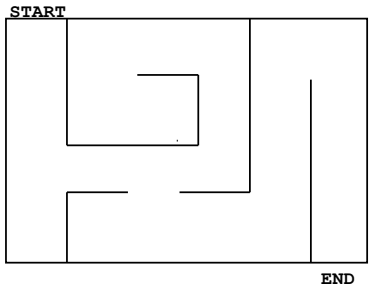
```
(derivative '(* 3 x x))  ->  '(* 6 x)

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

29

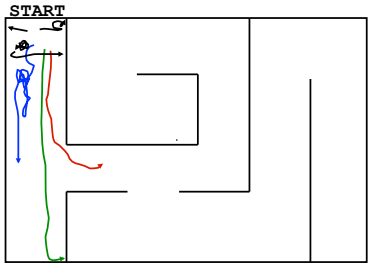
Genetic Programming

GP Example



30

Genetic Programming



31

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 ...))))
```

. . .

32