Lists

CSC 358/458
4.8.2003
Outline

- List
- Internals of Lists
- List operations
- Extended Example
Lists

- sequences of items
  - atoms or lists
- car
  - first element
- cdr
  - rest
- cons
  - builds a list
Other List Builders

- list
  - a list containing the arguments
- append
  - a list combining the arguments
Internals
(a b c)
Nested List
Improper List
What Does It Do?

- list
- append
It is easy to create lists that share structure

- (setf c (append a b))

Not a problem unless a or b is destructively modified
List Functions

- Accessors
  - navigating around in lists
- Operations
  - manipulate the whole list
- Destructive operations
  - modify the list
List Accessors

- car, cdr
- positional
  - first, second, third
  - (nth i lst)
  - (nthcdr i lst)
- end of the list
  - last
  - actually returns last cdr
List Operations

- Many!
- Mapping functions
- Boolean operations
- Finding
- Modifying (copy)
- Reduce
Mapping

- Functions that apply a function over and over across a list
- `(mapcar #'fn lst)`
  - applies the function to each element in the list
  - returns a list of the results
- Also works with multiple lists
  - `(mapcar #'fn lst1 lst2)`
Example

- dot product
- $v_1 \cdot v_2$
  - a vector containing the product of each dimension
Other Mapping Functions

- **maplist**
  - applies to successive cdrs

- **mapcan**
  - joins results with append

- **mapc, mapl**
  - like mapcar and maplist but does return results
Boolean Operations

- logical summary of a list
  - relative to a function
- (some #’fn lst)
  - true if any element of lst returns true when the function is applied
- every
  - false if any element returns false
- notany
- notevery
Finding Operations

- (member elem lst)
  - If elem is in list, returns the cdr headed by elem

- (position elem lst)
  - If elem is in list, return position index

- Both return nil is absent
Modifications

- (remove elem lst)
  - returns a copy of lst with elem removed
  - not “destructive”

- (substitute old new lst)
  - returns a copy of lst with old replaced by new
Reduce

- “reduce” list into a single value
- `(reduce #’fn lst)
  - applies fn to first two elements
  - applies fn to that result and next element
  - etc.
- `(reduce #’+ ‘(1 2 3 4)) => 10
Example

- polynomial evaluation
  - mapcar and reduce

- Original version

```
(defun evaluate (poly x)
  (if (null poly) 0
      (let ((term (car poly))
          (+ (* (term-factor term)
              (expt x (term-power term)))
            (evaluate (cdr poly) x))))
  ```
Equality Operators

- `=`
  - numerical equality only

- `eq`
  - reference equality

- `eql`
  - `eq`
  - numbers and characters

- `equal`
  - tests lists element by element
  - can be expensive!
Example

- defining equal
Most Lisp operations return "new" lists

- remove
- (setf lst '(b a b a))
- (remove 'a lst) => (b b b)
- lst => (b a b a)
- Use the result of the function

New cons cells created
Destructive Operations II

- Alter the contents of cons cells
  - rplaca = replace the car
  - rplacd = replace the cdr
  - nconc = destructive append
  - delete = destructive remove
  - nsubst = destructive subst

- Others
  - “n” means “non-consing”
Generalized Variables

- **setf**
  - can be similarly used
  - `(setf (car lst) val)`
  - `(setf (cdr lst) val)`

- Main reason to use
  - efficiency
Problems with Destructive Operations

- Shared structure
- Effects hard to predict
- Not necessary
- Main reason to use
  - efficiency
  - but remember Knuth
    - “Premature optimization is the root of all evil.”
Association List

- Allows lookup of values associated with keys
  - `((key1 . val1) (key2 . val2) ... (keyn . valn))`
- OK for short associations
  - large associations a hash table is better
- `(assoc key alist)`
  - returns the pair
- `(rassoc val alist)`
  - returns the pair
Example

- Multiple substitutions with sublis
Keyword parameters

- what if the value is not an atom?
  
  \[(\text{rassoc '}(21 \text{ M}) ')((\text{John} 21 \text{ M}) (\text{Jane} 19 \text{ F}) (\text{Joe} 25 \text{ M})))\]
  
  - doesn’t work because \text{rassoc} uses \text{eq}

- extra arguments to \text{rassoc}
  
  - \text{:test}
    
    - specifies which function to use instead of default
      
      \[(\text{rassoc '}(21 \text{ M}) ')((\text{John} 21 \text{ M}) (\text{Jane} 19 \text{ F}) (\text{Joe} 25 \text{ M})) :\text{test #'equal})\]
Keyword Parameters

- Many functions have these
- Most common
  - :test
  - :key
    - specifies a function to apply before testing
      (rassoc '21 '((John 21 M) (Jane 19 F) (Joe 25 M))
       :key #'car)
  - :start
  - :end
Defining Keyword Parameters

- `(defun foo (a b &key c (d 5))
  
  ...)
  
  5 is the default value for d

- can be called
  
  - (foo 1 2)
  
  - (foo 1 2 :c 1 :d 3)
Example

- sorting polynomials
- original version

(defun order-poly (poly)
  (sort poly #'compare-terms))

(defun compare-terms (term1 term2)
  (> (term-power term1) (term-power term2)))
Extended Example

- cryptography