More about list representation in LISP

✔ a list can be represented by
  ✗ the first or car of the list, and
  ✗ the rest or cdr of the list

✔ the first and rest of a list can each be a list

✔ This suggests using a binary tree to represent a list
  ✗ each internal node represents a list rooted at that node and has two children
    • the first of the list rooted at that node
    • the rest of the list rooted at that node
  ✗ each leaf of the tree is an atom

((a b) c)
list representation in LISP

✔ lists are sometimes represented with “box notation”

✔ box notation is a variant of usual binary tree notation

✔ internal nodes are represented by a box with two sub-boxes

![Image of box notation]

✔ the box represents a cons cell in memory

✔ the left sub-box represents a pointer to the first of the cons, the right sub-box represents a pointer to the rest of the cons

✔ if either is NIL, a slash is put in the sub-box

![Image of box notation with slash]
list representation in LISP

(a)

(a b)

(a)
list representation in LISP

```
((a b) c)
```

```
(a b)
```

```
(()
```
list representation

What is this list?

p. 234

bottom

winston & horn
creating and destroying lists

✔ When a list-creating function like list, cons, or append is called,...
✔ or when list structure is read in by the LISP system...
✔ cons cells are allocated from free storage

✔ When a cons cell is no longer being used, it is returned to free storage in a garbage collection process

✔ In pure functional LISP, no side effects exist to change the pointers in cons cells once they are allocated

✔ However, Common LISP provides many functions to change cons cell pointers: list surgery
list manipulation in LISP

✔ implementing append

(setf abc '(a b c) xyz '(x y z))

p. 238
top
winston & horn
list manipulation in LISP

✔ append involves copying cons cells to avoid unwanted side effects

\[(\text{setf abcxyz (append abc xyz))}\]

abcxyz
\((\text{A B C X Y Z})\)

abc
\((\text{A B C})\)

xyz
\((\text{X Y Z})\)

p. 239
top
winston & horn
list surgery in LISP

✔ nconc works like append except it performs list surgery instead of copying cons cells
✔ nconc produces side effects!

(setf abcxyz (nconc abc xyz))

abcxyz
(A B C X Y Z)

xyz
(X Y Z)

abc
(A B C X Y Z)

p. 239
bottom
winston & horn
list surgery in LISP

✔ many side-effect free list manipulation functions have a counterpart which performs list surgery

✔ the list-surgery version will be more efficient because it does not allocate new cons cells, but instead just changes pointers in existing cons cells

✔ however the list-surgery version will have side effects that may be hard to predict

✗ Example: delete works like remove, but performs list surgery. However the side effects of delete may not be what you would expect...

(setf tosses '(heads tails tails heads tails))
(HEADS TAILS TAILS HEADS TAILS)

(remove 'heads tosses)
(TAILS TAILS TAILS)

tosses
(HEADS TAILS TAILS HEADS TAILS)

(delete 'heads tosses)
(TAILS TAILS TAILS)

tosses
(HEADS TAILS TAILS TAILS)
list surgery functions

✔ usually the name of a function that does list surgery is the same as the corresponding side-effect-free version, preceded by “N”

✗ nreverse - reverse

✗ nunion - union

✗ nintersection - intersection

✗ etc.

✔ exceptions to this are

✗ nconc - append

✗ delete - remove
**do-it-yourself list surgery**

✔ setf can be used with list accessor functions to perform list surgery

✔ list surgery can produce circular lists

```
(setf fact3 '(circle))
(CIRCLE)

(setf (rest fact3) fact3)
```

✔ there are problems with printing out circular lists, and reclaiming cons cells in circular lists during garbage collection