ソフトウェア工学実験'93　課題2 (Lisp言語) ＃2の捕捉

久野　靖 *

1993.6.20

第2回の回答例が遅くなってすいません。遅巻きながらご参照ください。

練習問題の回答例

練習 1

```
>(setq x '(a b (c d) (e f (g h i))))
(A B (C D) (E F (G H I)))
>(car x)
A
>(cdr x)
(B (C D) (E F (G H I)))
>(caddr x)
(C D)
>(caaddr x)
C
>(cdaddr x)  ←ここで d と a が 4 つ (限界!)
(D)
>(car (cdaddr x)) ←これ以上は分ける
D
>(cdr (cdaddr x)) ←長さ 1 のリストの cdr は nil
NIL
>(caddr (cadddr x)) ←ちょっとやってみないと
(G H I)
>(cdaddr (cadddr x)) ←上のの cdr でよいわけだ。
(H I)
>(car (cadddr x)) ←最後のは「ひっかけ」。これと
E
>(cadr (cadddr x)) ←これをくっつける。
F
>(list (car (cadddr x)) (cadr (cadddr x)))
(E F)
```

最後のがちょっとエグかったという説もあるがまあ全部やんなくてもいいのですから。

練習 2

これもやるだけですが。練習になりましたか。

```
>(setq x '(a b c))
(A B C)
>(setq y '(d e))
(D E)
>(setq z 'z)
Z
>(append x y) ←連結は append
(A B C D E)
>(list x y)   ←並べたまま () をつけるのは list
((A B C) (D E))
>(cons x y)   ←先頭にくっつけるのは cons
((A B C) D E)
>(list (append x y)) ← () に入れるのはとにかく
list
((A B C D E))
>(cons z x)  ←先頭にくっつける
(Z A B C)
>(append (cons z x) (list z)) ←末尾につけるには
(Z A B C Z)                  ()  に 入 れ て か ら
append
>(append x (cons z y)) ←リストはそのまま append
(A B C Z D E)
```

練習 3

だんだん難しくなりましたか。

```
>(defun list3 (x) (list x x x))
LIST3
>(list3 'a)
(A A A)
>(defun list33 (x) (list3 (list3 x)))
LIST33
>(list33 'a)
((A A A) (A A A) (A A A))
>(defun addlist3 (l)
    (+ (car l) (cadr l) (caddr l)))
ADDLIST3
>(addlist3 '(1 2 3))
6
>(defun remlis3 (l)
    (list l (cdr l) (cddr l)))
REMLIS3
>(remlis3 '(a b c))
((A B C) (B C) (C))
```

*筑波大学大学院経営システム科学専攻

```
>(defun atob (x)
    (if (eq x 'a) 'b x))
ATOB
>(atob 'a)
B
>(atob 'c)
C
>(defun remtop (l)
    (if (null l) l (cdr l)))
REMTOP
>(remtop '(a b))
(B)
>(remtop '())
NIL
>(defun remtop1 (l)
    (if (null (cdr l)) l (cdr l)))
REMTOP1
>(remtop1 '(a))
(A)
>(remtop1 '(a b))
(B)
>(remtop1 '(a b c))
(B C)
>(defun atob3 (l)
    (list (atob (car l))
          (atob (cadr l))
          (atob (caddr l))))
ATOB3
>(atob3 '(c a t))
(C B T)
>(atob3 '(a n d))
(B N D)
```

```
        (cons x (repn x (- n 1)))))
REPN
>(repn 'z 6)
(Z Z Z Z Z Z)
>(defun listrev (l)
    (if (null l)
        nil
        (append (listrev (cdr l)) (list (car l)))))
LISTREV
>(listrev '(a b c d e))
(E D C B A)
>(defun shazou (l f)
    (if (null l)
        nil
        (cons (funcall f (car l)) (shazou (cdr l) f))))
SHAZOU
>(shazou '(1 2 3 4) #'-)
(-1 -2 -3 -4)
>(defun matome (l f)
    (if (null (cdr l))
        (car l)
        (funcall f (car l) (matome (cdr l) f))))
MATOME
>(matome '(1 2 3 4) #'+)
10
>(defun countatoms (l)
    (matome (shazou l #'(lambda (x) (if (atom x) 1 0)))
            #'+))
COUNTATOMS
>(countatoms '(a b (c d) (e (f g)) h (i) j))
4
```

練習 4

再帰関数には慣れましたか?

```
>(defun listsum (l)
    (if (null l)
        0
        (+ (car l) (listsum (cdr l)))))
LISTSUM
>(listsum '(3 4 5))
12
>(defun listtail (l)
    (if (null (cdr l))
        (car l)
        (listtail (cdr l))))
LISTTAIL
>(listtail '(d a c t))
T
>(listtail '(b a k e))
E
>(defun repn (x n)
    (if (= n 0)
        nil
```

練習 5

```
>(defun countallatoms (l)
    (cond ((null l) 0)
          ((atom l) 1)
          (t (+ (countallatoms (car l))
                (countallatoms (cdr l))))))
COUNTALLATOMS
>(countallatoms '(a b (c d) (e (f g)) h (i) j))
10
>(defun listrepl (l x y)
    (shazou l #'(lambda (z) (if (eq z x) y z))))
LISTREPL
>(listrepl '(e v e n) 'e 'x)
(X V X N)
>(defun remlisn (l)
    (cond ((null l) nil)
          (t (cons l (remlisn (cdr l))))))
REMLISN
>(remlisn '(a b c d))
((A B C D) (B C D) (C D) (D))
>(defun remove (l x)
    (cond ((null l) nil)
          ((eq (car l) x) (remove (cdr l) x))
          (t (cons (car l) (remove (cdr l) x)))))
Warning: REMOVE is being redefined.
REMOVE
>(remove '(e v e n) 'e)
(V N)
>(defun doreka (x l)
    (cond ((null l) nil)
          ((eq (car l) x) t)
          (t (doreka x (cdr l)))))
DOREKA
>(defun remset (l s)
    (cond ((null l) nil)
          ((doreka (car l) s) (remset (cdr l) s))
          (t (cons (car l) (remset (cdr l) s)))))
REMSET
>(remset '(e v e r y t h i n g) '(e t n))
(V R Y H I G)
>(defun replace (l x y)
    (cond ((null l) nil)
          ((atom l) (if (eq l x) y l))
          (t (cons (replace (car l) x y) (replace (cdr l) x y)))))
Warning: REPLACE is being redefined.
REPLACE
>(replace '(a (b c d) c e) 'c 'z)
(A (B Z D) Z E)
>(defun pair (x y)
    (cond ((null x) nil)
          (t (cons (list (car x) (car y)) (pair (cdr x) (cdr y))))))
PAIR
>(pair '(a b c) '(x y z))
((A X) (B Y) (C Z))
>(defun concat (x y)
    (cond ((null x) y)
          (t (cons (car x) (concat (cdr x) y)))))
CONCAT
>(concat '(a b c d) '(1 2 3))
(A B C D 1 2 3)
>(defun lookup (x l)
    (cond ((null l) nil)
          ((eq (caar l) x) (car l))
          (t (lookup x (cdr l)))))
LOOKUP
>(lookup 'a '((x y z) (a b c) (u v w)))
(A B C)
```