

File tree traversal Example

```
(defun dir (root filefn dirfn)
  (let ((files (directory root)))
    (loop for f in (reverse files) do
      (if (pathname-name f)
          (funcall filefn f)
          (funcall dirfn f)))))

(defun listdirs (root)
  (labels ((dirfn (root)
            (print root)
            (dir root
                 #'(lambda (f)
                     (declare (ignore f))
                     nil)
                 #'dirfn))))
    (dirfn root)))

(defun listall (root)
  (labels ((dirfn (root)
            (print root)
            (dir root
                 #'(lambda (f) (print f))
                 #'dirfn))))
    (dirfn root)))

(defun recurse (root filefn dirfn)
  (labels ((dirfn (root)
            (funcall dirfn root)
            (dir root
                 filefn
                 #'dirfn))))
    (dirfn root)))
```

Output

```
* (listall #p"../")
```

```
#p"../"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/x.pl~"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/x.pl"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/filter.pl"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/concordance.pl~"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/concordance.pl"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/lisp/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/lisp/examples.sparcf"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/lisp/examples.lisp~"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/lisp/examples.lisp"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/flex/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/flex/pascal.l~"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/flex/lex.yy.c"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/flex/a.out"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/bison/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/bison/tests/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/bison/calc/a.out"
```

```
NIL
```

```
* (listdirs #p"../")
```

```
#p"../"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/perl/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/lisp/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/flex/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/bison/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/bison/tests/"
```

```
#p"/home/fac/swm/PLC/Lecture/tmp/bison/calc/"
```

```
NIL
```

Functions

```
(defun square (x)
  (* x x))
```

```
(square 6)
```

```
(square (square 3))
```

```
(defun app (x y)
  (cond ((null x)
        y)
        (t (cons (car x)
                  (append (cdr x) y)))))
```

```
* (app '(a b) '(c d))
```

```
(A B C D)
```

```
* (app '(a b) 'c)
```

```
(A B . C)
```

```
* (app 'x 'y)
```

```
Type-error in KERNEL::OBJECT-NOT-LIST-ERROR-HANDLER: X
is not of type LIST
```

```
Restarts:
```

```
0: [ABORT] Return to Top-Level.
```

```
Debug (type H for help)
```

```
(CAR 1 X) [:EXTERNAL]
```

```
0] :q
```

Copying

```
(defun shallow-copy (lst)
  (cond ((null lst)
         nil)
        (t (cons (car lst)
                  (shallow-copy (cdr lst))))))
```

```
* (shallow-copy '(x y z))
```

```
(X Y Z)
```

```
(let ((x '(a b c)))
  (let ((y (shallow-copy x)))
    (eq x y)))
```

NIL

```
(defun deep-copy (lst)
  (cond ((atom lst)
         lst)
        (t (cons (deep-copy (car lst))
                  (deep-copy (cdr lst))))))
```

```
(let ((x '((a) b c)))
  (let ((y (shallow-copy x)))
    (eq (car x) (car y))))
```

T

```
(let ((x '((a) b c)))
  (let ((y (deep-copy x)))
    (eq (car x) (car y))))
```

NIL

Reversing a list

```
(defun rev (lst)
  (rev1 lst nil))
```

```
(defun rev1 (lst1 lst2)
  (cond ((null lst1)
         lst2)
        (t (rev1 (cdr lst1)
                  (cons (car lst1) lst2)))))
```

```
* (rev '(a b c d))
```

```
(D C B A)
```

Mapping

```
* (mapcar #'cons '(1 2) '(a b))
```

```
((1 . A) (2 . B))
```

```
* (mapcar #'list '(1 2) '(a b))
```

```
((1 A) (2 B))
```

```
* (mapcar #'(lambda (x) (+ x 1)) '(1 2 3))
```

```
(2 3 4)
```

```
(defun mc (fn lst1 lst2)
  (cond ((and (not (null lst1))
              (not (null lst2)))
         (cons (funcall fn (car lst1) (car lst2))
                (mc fn (cdr lst1) (cdr lst2))))
        (t nil)))
```

Generating Permutations

```
(defun skip (lst skp)
  (cond ((eq lst skp)
        (cdr lst))
        (t (cons (car lst) (skip (cdr lst) skp)))))

(defun permute (lst fn)
  (cond ((null lst)
        (funcall fn lst))
        (t (mapl #'(lambda (x)
                     (let ((tail (skip lst x))
                           (head (car x)))
                       (permute tail
                                #'(lambda (taillist)
                                    (funcall fn
                                             (cons
                                              head
                                              taillist))))))
                     lst)))
        (nil)

* (permute '(a b c) #'print)
```

```
(A B C)
(A C B)
(B A C)
(B C A)
(C A B)
(C B A)
```

NIL

*

Factorial

```
(defun fact (n)
  (if (zerop n)
      1
      (* n (fact (- n 1)))))
```

```
* (fact 10)
```

3628800

```
* (fact 100)
```

```
9332621544394415268169923885626670049071596826438162146
8592963895217599993229915608941463976156518286253697920
827223758251185210916864000000000000000000000000000000
```

```
* (fact 1000)
```

```
4023872600770937735437024339230039857193748642107146325
4379991042993851239862902059204420848696940480047998861
0197196058631666872994808558901323829669944590997424504
0870737599188236277271887325197795059509952761208749754
6249704360141827809464649629105639388743788648733711918
1045825783647849977012476632889835955735432513185323958
4630755574091142624174743493475534286465766116677973966
6882029120737914385371958824980812686783837455973174613
6085379534524221586593201928090878297308431392844403281
2315586110369768013573042161687476096758713483120254785
8932076716913244842623613141250878020800026168315102734
1827977704784635868170164365024153691398281264810213092
7612448963599287051149649754199093422215668325720808213
3318611681155361583654698404670897560290095053761647584
7728421889679646244945160765353408198901385442487984959
9533191017233555566021394503997362807501378376153071277
6192684903435262520001588853514733161170210396817592151
0907788019393178114194545257223865541461062892187960223
8389714760885062768629671466746975629112340824392081601
5378088989396451826324367161676217916890977991190375403
1274622289988005195444414282012187361745992642956581746
```


Generating partitions

```
(defun partition (lst)
  (cond ((null lst)
        (list nil))
        (t (let* ((head (car lst))
                  (tail (cdr lst))
                  (tailpart (partition tail)))
              (append (mapcar #'(lambda (part)
                                (cons (list head)
                                      part))
                        tailpart)
                      (mapcan
                       #'(lambda (part)
                           (maplist
                            #'(lambda (piece1)
                                (maplist
                                 #'(lambda (piece2)
                                     (if (eq piece1
                                             piece2)
                                         (cons head
                                               (car piece2))
                                         (car piece2))))
                                part))
                          part))
                       tailpart))))))
```

Output

* (partition ' (a))

((A))

* (partition ' (a b))

((A) (B))

((A B))

* (partition ' (a b c))

((A) (B) (C))

((A) (B C))

((A B) (C))

((B) (A C))

((A B C))

* (partition ' (a b c d))

((A) (B) (C) (D))

((A) (B) (C D))

((A) (B C) (D))

((A) (C) (B D))

((A) (B C D))

((A B) (C) (D))

((B) (A C) (D))

((B) (C) (A D))

((A B) (C D))

((B) (A C D))

((A B C) (D))

((B C) (A D))

((A C) (B D))

((C) (A B D))

((A B C D))