```
 1: (* $Id: hof.ml,v 361.7 2008-03-06 18:14:34-08 - - $ *)
 2:
 3: let compose f g x = f (g x)
 4:
 5: let compose2 f g x y = f (g x y)
 6:
 7: let swap f x y = f y x
 8:
 9: let even x = x mod 2 = 0
10:
11: let odd = compose not even
12:
13: let cons car cdr = car::cdr
14:
15: let rec foldl fn unit list = match list with
16:     | [] -> unit
17:     | car::cdr -> foldl fn (fn unit car) cdr
18:
19: let rec foldr fn unit list = match list with
20:     | [] -> unit
21:     | car::cdr -> fn car (foldr fn unit cdr)
22:
23: let rec foldl2 fn unit list1 list2 = match list1, list2 with
24:     | [], [] -> unit
25:     | car1::cdr1, car2::cdr2 -> foldl2 fn (fn unit car1 car2) cdr1 cdr2
26:     | _, _ -> raise (Invalid_argument "foldl2")
27:
28: let rec foldr2 fn unit list1 list2 = match list1, list2 with
29:     | [], [] -> unit
30:     | car1::cdr1, car2::cdr2 -> fn car1 car2 (foldr2 fn unit cdr1 cdr2)
31:     | _, _ -> raise (Invalid_argument "foldl2")
32:
33: let sum = foldl (+) 0
34:
35: let lengthrec list =
36:     let rec lengthrec' list' len' = match list' with
37:         | [] -> len'
38:         | _::cdr -> lengthrec' cdr (len' + 1)
39:     in  lengthrec' list 0
40:
41: let lengthf list = foldl (fun len _ -> len + 1) 0 list
42:
43: let reverserec list =
44:     let rec reverserec' list' revlist = match list' with
45:         | [] -> revlist
46:         | car::cdr -> reverserec' cdr (car::revlist)
47:     in  reverserec' list []
48:
49: let reversef list = foldl (swap cons) [] list
50:
51: let filterrec test list =
52:     let rec filterrec' list' = match list' with
53:         | [] -> []
54:         | car::cdr when test car -> car :: (filterrec' cdr)
55:         | _::cdr -> filterrec' cdr
56:     in  filterrec' list
57:
58: let filterf test =
```

```
59:     foldr (fun car cdr -> if test car then car::cdr else cdr) []
60:
61: let maprec fn list =
62:     let rec maprec' list' = match list' with
63:         | [] -> []
64:         | car::cdr -> (fn car) :: maprec' cdr
65:     in  maprec' list
66:
67: let mapf fn = foldr (compose cons fn) []
68:
69: let mapf2 fn = foldr2 (compose2 cons fn) []
70:
71: let innerprodl = foldl2 (fun sum val1 val2 -> sum + val1 * val2) 0
72:
73: let innerprodr = foldr2 (compose2 (+) ( * )) 0
74:
75: let rec memberrec elt list = match list with
76:     | [] -> false
77:     | car::_ when car = elt -> true
78:     | _::cdr -> memberrec elt cdr
79:
80: let memberf elt = foldl (fun car cdr -> car = elt || cdr) false
81:
82: let zipf list1 list2 = mapf2 (fun a b -> a, b) list1 list2
83:
84: let qsort (<?) list =
85:     let (>=?) x y = not (x <? y) in
86:     let rec qsort' list' = match list' with
87:         | [] -> []
88:         | car::cdr ->
89:             qsort' (filterf (swap (<?) car) cdr)
90:           @ [car]
91:           @ qsort' (filterf (swap (>=?) car) cdr)
92:     in  qsort' list
93:
```