```
 1: (* $Id: lazythunk-p4.ml,v 353.1 2005-05-17 19:25:45-07 - - $ *)
 2:
 3: open Printf
 4:
 5: (* re-implementation of module Lazy *)
 6:
 7: #load "camlp4o.cma";;
 8: let lazyexpand _ expr = "(ref (Delay (fun () -> (" ^ expr ^ "))))";;
 9: Quotation.add "lazy" (Quotation.ExStr lazyexpand);;
10:
11: type 'a suspension =
12:     | Value of 'a
13:     | Excep of exn
14:     | Delay of (unit -> 'a)
15:
16: type 'a thunk = 'a suspension ref
17:
18: exception Thunk_cycle
19:
20: let rec force thunk = match !thunk with
21:     | Delay delay -> (thunk := Excep Thunk_cycle;
22:                       try  let value = delay ()
23:                            in  (thunk := Value value; value)
24:                       with excep -> (thunk := Excep excep; raise excep))
25:     | Value value -> value
26:     | Excep excep -> raise excep
27:
28: let (!?) = force
29:
30: (* stream and lazy stuff *)
31:
32: type 'a stream = End | Stream of 'a * 'a stream thunk
33:
34: exception End_stream
35:
36: let (@::) hd tl = Stream (hd, tl)
37:
38: let head = function
39:     | End            -> raise End_stream
40:     | Stream (hd, _) -> hd
41:
42: let tail = function
43:     | End            -> raise End_stream
44:     | Stream (_, tl) -> !?tl
45:
46: let rec take n stream = match n, stream with
47:     | _, End             -> End
48:     | n, _ when n <= 0   -> End
49:     | _, Stream (hd, tl) -> Stream (hd, <:lazy< (take (n - 1) !?tl)>>)
50:
51: let rec drop n stream = match n, stream with
52:     | _, End             -> End
53:     | n, _ when n <= 0   -> stream
54:     | _, Stream (hd, tl) -> drop (n - 1) !?tl
55:
56: let rec list_of_stream = function
57:     | End             -> []
58:     | Stream (hd, tl) -> hd :: list_of_stream !?tl
```

```
 59:
 60: let rec iter fn1 = function
 61:     | End             -> ()
 62:     | Stream (hd, tl) -> (fn1 hd; iter fn1 !?tl)
 63:
 64: let rec iter2 fn2 = function
 65:     | End, _ -> ()
 66:     | _, End -> ()
 67:     | Stream (hd1, tl1), Stream (hd2, tl2)
 68:             -> (fn2 hd1 hd2; iter2 fn2 !?tl1 !?tl2)
 69:
 70: let rec iter3 fn3 = function
 71:     | End, _, _ -> ()
 72:     | _, End, _ -> ()
 73:     | _, _, End -> ()
 74:     | Stream (hd1, tl1), Stream (hd2, tl2), Stream (hd3, tl3)
 75:                -> (fn3 hd1 hd2 hd3; iter3 fn3 !?tl1 !?tl2 !?tl3)
 76:
 77: let rec zip fn = function
 78:     | End, _ -> End
 79:     | _, End -> End
 80:     | Stream (hd1, tl1), Stream (hd2, tl2)
 81:             -> Stream (fn hd1 hd2, <:lazy< (zip fn !?tl1 !?tl2)>>)
 82:
 83: (* stuff that uses streams and Nums *)
 84:
 85: let rec range head limit =
 86:     if head > limit
 87:     then End
 88:     else let next = head + 1
 89:          in  Stream (head, <:lazy< (range next limit)>>)
 90:
 91: let naturals = range 0 max_int
 92:
 93: let fac n =
 94:     let rec fac' n m = match n with
 95:         | 0 -> m
 96:         | n -> fac' (n - 1) (n * m)
 97:     in  if n < 0 then invalid_arg "fac"
 98:                  else fac' n 1
 99:
100: let printfac n = printf "%d! = %d\n" n (fac n)
101:
102: let printfacs n = iter printfac (take n naturals)
103:
104: (* let fib = 0 : 1 : zip (+) fib (tail fib) *)
105:
106: let fibstream =
107:     let rec fibstream0 = Stream (0, fibstream1)
108:         and fibstream1 = <:lazy< (Stream (1, fibstream2))>>
109:         and fibstream2 = <:lazy< (zip (+) fibstream0 !?fibstream1)>>
110:     in  fibstream0
111:
112: let printfib n nfib nfib' =
113:     printf "fib(%3d) = %11d, %20.15f\n"
114:            n nfib (float_of_int nfib /. float_of_int nfib')
115:
116: let printfibs n = iter3 printfib naturals
```

```
117:                                    (take n fibstream)
118:                                    (take n (drop 1 fibstream))
119:
```