```
 1: (* $Id: lazythunk.ml,v 361.1 2006-03-02 18:48:23-08 - - $ *)
 2:
 3: open Printf
 4:
 5: (* re-implementation of module Lazy *)
 6:
 7: type 'a promise =
 8:     | Value of 'a
 9:     | Excep of exn
10:     | Delay of (unit -> 'a)
11:
12: type 'a thunk = 'a promise ref
13:
14: let force thunk = match !thunk with
15:     | Value value -> value
16:     | Excep excep -> raise excep
17:     | Delay delay -> (try  let value = delay ()
18:                            in  (thunk := Value value; value)
19:                       with excep -> (thunk := Excep excep;
20:                                      raise excep))
21:
22: let (!!) = force
23:
24: (* stream and lazy stuff *)
25:
26: type 'a stream = End | Stream of 'a * 'a stream thunk
27:
28: exception End_stream
29:
30: let (@::) car cdr = Stream (car, cdr)
31:
32: let head stream = match stream with
33:     | End -> raise End_stream
34:     | Stream (car, _) -> car
35:
36: let tail stream = match stream with
37:     | End -> raise End_stream
38:     | Stream (_, cdr) -> !!cdr
39:
40: let rec take n stream = match n, stream with
41:     | _, End -> End
42:     | n, _ when n <= 0 -> End
43:     | _, Stream (car, cdr) ->
44:          Stream (car, ref (Delay (fun () -> take (n - 1) !!cdr)))
45:
46: let rec list_of_stream stream = match stream with
47:     | End -> []
48:     | Stream (car, cdr) -> car :: list_of_stream !!cdr
49:
50: let rec drop n stream = match n, stream with
51:     | _, End -> End
52:     | n, _ when n <= 0 -> stream
53:     | _, Stream (car, cdr) -> drop (n - 1) !!cdr
54:
55: let rec iter fn stream = match stream with
56:     | End -> ()
57:     | Stream (car, cdr) -> (fn car; iter fn !!cdr)
58:
```

```
 59: let rec iter2 fn stream1 stream2 = match stream1, stream2 with
 60:     | End, _ -> ()
 61:     | _, End -> ()
 62:     | Stream (car1, cdr1), Stream (car2, cdr2)
 63:          -> (fn car1 car2; iter2 fn !!cdr1 !!cdr2)
 64:
 65: let rec iter3 fn stream1 stream2 stream3 =
 66:     match stream1, stream2, stream3 with
 67:     | End, _, _ -> ()
 68:     | _, End, _ -> ()
 69:     | _, _, End -> ()
 70:     | Stream (car1, cdr1), Stream (car2, cdr2), Stream (car3, cdr3)
 71:          -> (fn car1 car2 car3; iter3 fn !!cdr1 !!cdr2 !!cdr3)
 72:
 73: let rec map2 fn stream1 stream2 = match stream1, stream2 with
 74:     | End, _ -> End
 75:     | _, End -> End
 76:     | Stream (car1, cdr1), Stream (car2, cdr2)
 77:          -> Stream (fn car1 car2,
 78:                     ref (Delay (fun () -> map2 fn !!cdr1 !!cdr2)))
 79:
 80: (* stuff that uses streams and Nums *)
 81:
 82: let rec range head limit =
 83:     if head > limit
 84:     then End
 85:     else let next = head + 1
 86:          in  Stream (head, ref (Delay (fun () -> range next limit)))
 87:
 88: let naturals = range 0 max_int
 89:
 90: let fac n =
 91:     let rec fac' n m = match n with
 92:         | 0 -> m
 93:         | n -> fac' (n - 1) (n * m)
 94:     in  if n < 0 then invalid_arg "fac"
 95:                  else fac' n 1
 96:
 97: let printfac n = printf "%2d! = %10d\n" n (fac n)
 98:
 99: let printfacs n = iter printfac (take n naturals)
100:
101: (* lazy let fib = 0 : 1 : map2 (+) fib (tail fib) *)
102:
103: let fibstream =
104:     let rec stream0 = Stream (0, stream1)
105:         and stream1 = ref (Delay (fun () -> Stream (1, stream2)))
106:         and stream2 = ref (Delay (fun () -> map2 (+) stream0 !!stream1))
107:     in  stream0
108:
109: let printfib n nfib nfib' =
110:     printf "fib(%3d) = %11d, %20.15f\n"
111:            n nfib (float_of_int nfib /. float_of_int nfib')
112:
113: let printfibs n = iter3 printfib naturals
114:                                 (take n fibstream)
115:                                 (take n (drop 1 fibstream))
116:
```

```
 1: type 'a promise = Value of 'a | Excep of exn | Delay of (unit -> 'a)
 2: type 'a thunk = 'a promise ref
 3: val force : 'a promise ref -> 'a
 4: val ( !! ) : 'a promise ref -> 'a
 5: type 'a stream = End | Stream of 'a * 'a stream thunk
 6: exception End_stream
 7: val ( @:: ) : 'a -> 'a stream thunk -> 'a stream
 8: val head : 'a stream -> 'a
 9: val tail : 'a stream -> 'a stream
10: val take : int -> 'a stream -> 'a stream
11: val list_of_stream : 'a stream -> 'a list
12: val drop : int -> 'a stream -> 'a stream
13: val iter : ('a -> 'b) -> 'a stream -> unit
14: val iter2 : ('a -> 'b -> 'c) -> 'a stream -> 'b stream -> unit
15: val iter3 :
16:   ('a -> 'b -> 'c -> 'd) -> 'a stream -> 'b stream -> 'c stream -> unit
17: val map2 : ('a -> 'b -> 'c) -> 'a stream -> 'b stream -> 'c stream
18: val range : int -> int -> int stream
19: val naturals : int stream
20: val fac : int -> int
21: val printfac : int -> unit
22: val printfacs : int -> unit
23: val fibstream : int stream
24: val printfib : int -> int -> int -> unit
25: val printfibs : int -> unit
```