```
 1: (* $Id: lazystream.ml,v 353.2 2005-05-17 19:32:12-07 - - $ *)
 2:
 3: let (!!) = Lazy.force
 4:
 5: let prtf = Printf.printf
 6:
 7: (* stream and lazy stuff *)
 8:
 9: type 'a stream = End | Stream of 'a * 'a stream Lazy.t
10:
11: let (@::) car cdr = Stream (car, cdr)
12:
13: let head stream = match stream with
14:     | End -> invalid_arg "head"
15:     | Stream (car, _) -> car
16:
17: let tail stream = match stream with
18:     | End -> invalid_arg "tail"
19:     | Stream (_, cdr) -> !!cdr
20:
21: let rec take n stream = match n, stream with
22:     | 0, _ when n <= 0 -> End
23:     | _, End -> End
24:     | _, Stream (car, cdr) -> Stream (car, lazy (take (n - 1) !!cdr))
25:
26: let rec list_of_stream stream = match stream with
27:     | End -> []
28:     | Stream (car, cdr) -> car :: list_of_stream !!cdr
29:
30: let rec iter fn stream = match stream with
31:     | End -> ()
32:     | Stream (car, cdr) -> (fn car; iter fn !!cdr)
33:
34: let rec iter2 fn stream1 stream2 = match stream1, stream2 with
35:     | End, _ -> ()
36:     | _, End -> ()
37:     | Stream (car1, cdr1), Stream (car2, cdr2)
38:             -> (fn car1 car2; iter2 fn !!cdr1 !!cdr2)
39:
40: let rec map2 fn stream1 stream2 = match stream1, stream2 with
41:     | End, _ -> End
42:     | _, End -> End
43:     | Stream (car1, cdr1), Stream (car2, cdr2)
44:             -> Stream (fn car1 car2, lazy (map2 fn !!cdr1 !!cdr2))
45:
46: (* stuff that uses streams and Nums *)
47:
48: let rec range head limit =
49:     if head > limit
50:     then End
51:     else let next = head + 1
52:          in  Stream (head, lazy (range next limit))
53:
54: let naturals = range 0 max_int
55:
56: let fac n =
57:     let rec fac' n m = match n with
58:         | 0 -> m
```

```
59:          | n -> fac' (n - 1) (n * m)
60:      in  if n < 0 then invalid_arg "fac"
61:                   else fac' n 1
62:
63: let printfac n = prtf "%d! = %d\n" n (fac n)
64:
65: let printfacs n = iter printfac (take n naturals)
66:
67: (* let fib = 0 : 1 : map2 (+) fib (tail fib) *)
68:
69: let fibstream =
70:     let rec stream0 = Stream (0, stream1)
71:         and stream1 = lazy (Stream (1, stream2))
72:         and stream2 = lazy (map2 (+) stream0 !!stream1)
73:     in  stream0
74:
75: let printfib n nfib = prtf "fib(%d) = %d\n" n nfib
76:
77: let printfibs n = iter2 printfib naturals (take n fibstream)
78:
```

```
 1: val ( !! ) : 'a Lazy.t -> 'a
 2: val prtf : ('a, out_channel, unit) format -> 'a
 3: type 'a stream = End | Stream of 'a * 'a stream Lazy.t
 4: val ( @:: ) : 'a -> 'a stream Lazy.t -> 'a stream
 5: val head : 'a stream -> 'a
 6: val tail : 'a stream -> 'a stream
 7: val take : int -> 'a stream -> 'a stream
 8: val list_of_stream : 'a stream -> 'a list
 9: val iter : ('a -> 'b) -> 'a stream -> unit
10: val iter2 : ('a -> 'b -> 'c) -> 'a stream -> 'b stream -> unit
11: val map2 : ('a -> 'b -> 'c) -> 'a stream -> 'b stream -> 'c stream
12: val range : int -> int -> int stream
13: val naturals : int stream
14: val fac : int -> int
15: val printfac : int -> unit
16: val printfacs : int -> unit
17: val fibstream : int stream
18: val printfib : int -> int -> unit
19: val printfibs : int -> unit
```