```
 1: signature QUEUE =
 2:     sig
 3:         type 'a queue ;
 4:         val makeEmpty : unit -> 'a queue ;
 5:         val enqueue : ('a * 'a queue) -> 'a queue ;
 6:         val dequeue : 'a queue -> ('a * 'a queue) option ;
 7:         val isEmpty : 'a queue -> bool ;
 8:     end ;
 9:
10: structure Queue : QUEUE =
11: struct
12:
13: datatype 'a queue = Q of { head   : 'a list,
14:                            worker : 'a list,
15:                            tail   : 'a list } ;
16: (* invariant:
17:  * length head = length worker + length tail
18:  *)
19:
20: fun makeEmpty () = Q { head=[],worker=[],tail=[]} ;
21: fun isEmpty (Q x) = null (#head x) ;
22:
23: fun singleton a = let val list1 = [a] in Q { head=list1,worker=list1,tail=[]} ;
24:
25: fun enqueue (elt,Q q) = rebuild (#head q, #worker q, elt :: #tail q)
26:
27: and dequeue (Q q) = case #head q of
28:                          [] => NONE
29:                        | h::hs => SOME(
30:                                   h,
31:                                   rebuild (hs, #worker q, #tail q))
32:
33: (* call invariant:
34:  * length head + 1 = length worker + length tail
35:  *)
36: and rebuild (head,worker,tail)
37:   = case worker of
38:         w::worker' => Q{head=head,worker=worker',tail=tail}
39:       | [] => let
40:             (* key assertion: head has no lazyness left *)
41:             val newHead = rotate(head,tail,[])
42:         in Q{head=newHead,worker=newHead,tail=[])
43:
44: (* call invariant:
45:  * length 1st arg + 1 = length 2nd arg
46:  *)
47: and rotate ([],r::rTail,acc) = r::acc
48:   | rotate (h::head,r::rTail,acc) = h::rotate(head,rTail,r::acc)
49:
50: end ;
```