Exercise 3.21.

Ben Bitdiddle decides to test the queue implementation described above. He types in the procedures to the Lisp interpreter and proceeds to try them out:

```
(define q1 (make-queue))

(insert-queue! q1 'a)
((a) a)

(insert-queue! q1 'b)
((a b) b)

(delete-queue! q1)
((b) b)

(delete-queue! q1)
(() b)
```

"It's all wrong!" he complains. "The interpreter's response shows that the last item is inserted into the queue twice. And when I delete both items, the second `b` is still there, so the queue isn't empty, even though it's supposed to be." Eva Lu Ator suggests that Ben has misunderstood what is happening. "It's not that the items are going into the queue twice," she explains. "It's just that the standard Lisp printer doesn't know how to make sense of the queue representation. If you want to see the queue printed correctly, you'll have to define your own print procedure for queues." Explain what Eva Lu is talking about. In particular, show why Ben's examples produce the printed results that they do. Define a procedure `print-queue` that takes a queue as input and prints the sequence of items in the queue.

Answer.

We know that a queue is represented as a pair of pointers, that is, `front-ptr` and `rear-ptr`, which indicate, respectively, the first and last pairs in an ordinary list. Also notice that when the standard Lisp printer encounters a pair, it prints the `car` of that pair as well as the `cdr` of it. Hence, printing a queue via the standard Lisp printer brings the user a sequence of all the elements in a queue together with its rare element. This explains why the output puzzles Ben a lot.

To make the output more intuitive, we can define a procedure `print-queue` that display only the sequence of all the elements in a queue, that is, the sequence designated by `front-ptr` of a queue:

```
(define (print-queue queue)
  (front-ptr queue))
```