Exercise 4.73.

Why does `flatten-stream` use `delay` explicitly? What would be wrong with defininig it as follows:

```
(define (flatten-stream stream)
  (if (stream-null? stream)
      the-empty-stream
      (interleave
       (stream-car stream)
       (flatten-stream (stream-cdr stream)))))
```

Answer.

Like the `integral` procedure in section 3.5.4, `flatten-stream` employs `delay` explicitly to modify the order of evaluation, in that we can generate part of the answer given only partial information about the arguments. The `delay` operation supresses the evaluation of subproblem `flatten-stream` generates. It enables the `interleave-delayed` procedure to combine the first element in its first argument into the final output stream in advance, even though the second argument has not been evaluated. This strategy postpones looping in the presence of infinite streams.

Conversely, by leaving out `delay`, as the variation suggested in the problem for example, `flatten-stream` will loop eternally and produce nothing in the presence of infinite stream. By the order of evaluation in Scheme, the `interleave` procedure won't set out to combine the final output stream until both its arguments get evaluated. However, evaluating the second argument causes `flatten-stream` unfold into its subproblem eternally in the presence of infinite stream, for the recursive boundary can never be reached.