First, there is the issue of correctness. The ease of writing potentially infinite programs may mask the difficulty of ensuring that they will always terminate. We have seen that `within` assumes a precondition, not stated in its presentation; this precondition, requiring that elements decrease to below `eps`, cannot be finitely evaluated on an infinite sequence (it is semi-decidable). These are tricky techniques for designers to use, as illustrated by the problem of how many lazy functional programmers it takes to change a light bulb. (It is hard to know in advance. If there are any lazy functional programmers left, ask one to change the bulb. If she fails, try the others.)

Second and last, lazy manipulation of infinite structures is possible in a nonfunctional design environment, without any special language support. The abstract data type approach (also known as object-oriented design) provides the appropriate solution. Finite sequences and lists in Eiffel libraries are available through an API relying on a notion of "cursor" (see Figure 13-2).

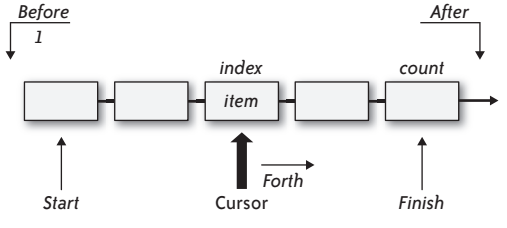

FIGURE 13-2. Cursors in Eiffel lists

Commands to move the cursor are `start` (go to first item), `forth` (move to next item), and `finish`. Boolean queries `before` and `after` tell if the cursor is before the first element or after the last. If neither holds, `item` returns the element at cursor position, and `index` its index.

It is easy to adapt this specification to cover infinite sequences: just remove `finish` and `after` (as well as `count`, the number of items). This is the specification of the deferred (abstract) class `COUNTABLE` in the Eiffel library. Some of its descendants include `PRIMES`, `FIBONACCI`, and `RANDOM`; each provides its implementations of `start`, `forth`, and `item` (plus, in the last case, a way to set the seed for the pseudo-random number generator). To obtain successive elements of one of these infinite sequences, it suffices to apply `start` and then query for `item` after any finite number of applications of `forth`.

Any infinite sequential structure requiring finite evaluation can be modeled in this style. Although this does not cover all applications of lazy evaluation, the advantage is to make the infinite structure explicit, so that it is easier to establish the correctness of a lazy computation.

State Intervention

The functional approach seeks to rely directly on the properties of mathematical functions by rejecting the assumption, present but implicit in imperative approaches, that computing