```
        end
end
```

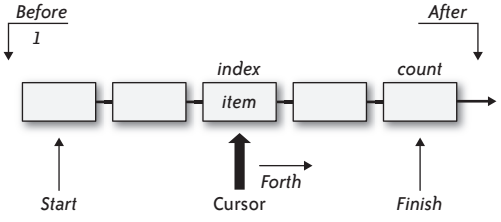

FIGURE 13-4. A polymorphic list with cursors

This applies the operations of class `LIST` directly to a `COMPOSITE_PART`, since the latter class inherits from the former. The result of `item` can be of any of the descendant types of `PUDDING`; since it may as a consequence denote objects of several types, it is known as a *polymorphic variable* (more precisely in this case, a polymorphic query). An entire `COMPOSITE_PART` structure, containing items of different types, is known as a *polymorphic container*. Polymorphic containers are made possible by the combination of polymorphism, itself resulting from inheritance, and genericity. (As these are two very different mechanisms, the functional programming term "parametric polymorphism" for genericity can cause confusion.)

The polymorphism of `item` implies that successive executions of the call `item.sugar_content` will typically apply to objects of different types; the corresponding classes may have different versions of the query `sugar_content`. *Dynamic binding* is here the guarantee that such calls will in each case apply the appropriate version, based on the type of the object actually attached to `item`. In the case of a part that is itself composite, this will be the above version, applied recursively; but it could be any other—for example, the version for `CREAM`.

Here as in most current approaches to OO design, polymorphism is controlled by the type system. The type of `item`'s value is variable, but only within descendants of `PUDDING` as specified by the generic parameter of `COMPOSITE_PART`. This is part of a development that has affected both the functional programming world and the object-oriented world: using increasingly sophisticated type systems (still based on a small number of simple concepts such as inheritance, genericity, and polymorphism) to embody a growing part of the intelligence about a system's architecture into its type structure.

Deferred Classes and Features

Classes `PUDDING` and `PUDDING_PART` are marked as "deferred" (with an asterisk in the BON object-oriented modeling notation [Walden and Nerson 1994]) in the earlier diagram of the class structure. This means they are not completely implemented; another term is "abstract class." A deferred class will generally have deferred *features*, possessing a signature and (importantly) a contract, but no implementation. Implementations appear in nondeferred ("effective")