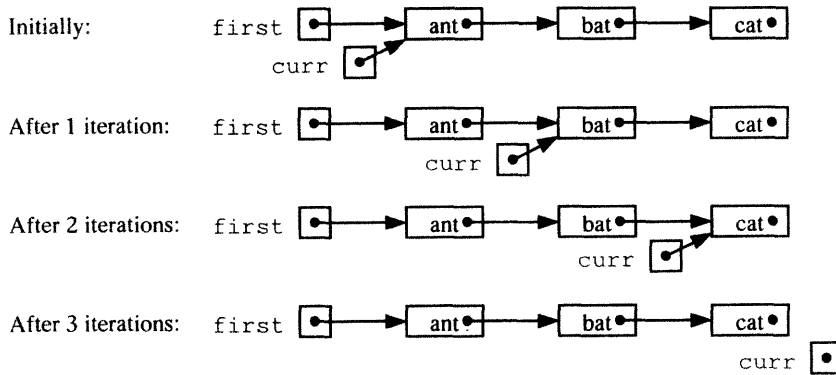


The local variable `curr` (an abbreviation of ‘current’) is made to refer to each node of the SLL in turn. Initially it refers to the first node (`curr = first`). Each iteration of the loop accesses the element in the node that `curr` refers to (`curr.element`), and then updates `curr` to refer to that node’s successor (`curr = curr.succ`). Iteration continues as long as the end of the SLL has not been reached (`curr != null`).

A typical call to this method would be:

```
zoo1.printFirstToLast();
```

where `zoo1` is the SLL object created in Example 4.1. The following diagram shows the method’s progress:



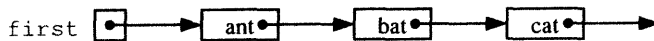
The printed output should be ‘ant bat cat’.

If the SLL is empty, `first` is null, and the method prints nothing at all.

Example 4.2 illustrated traversal of an SLL. In general, *traversal* of a data structure means visiting some or all of its nodes in some predetermined order. It is straightforward to traverse an SLL in first-to-last order.

EXAMPLE 4.3 SLL manipulation

Consider an SLL whose nodes contain ‘ant’, ‘bat’, ‘cat’, etc:



Each of the following code fragments manipulates the above SLL’s structure in a different way. Assume that `first` is a link to the SLL’s first node.

(a) The following code fragment deletes the SLL’s first node:

```
first = first.succ;
```