

DLL has at least three nodes):

```

DLLNode secondLast = last.pred;
DLLNode thirdLast = secondLast.pred;
thirdLast.succ = last;
last.succ = secondLast;
secondLast.succ = null;
last.pred = thirdLast;
secondLast.pred = last;
last = secondLast;
    
```

Comparing Example 4.6 with Example 4.3, we see that we can restructure a DLL in much the same way as we restructure an SLL, except that we must update the predecessor links as well as the successor links. Therefore, even greater care is needed to update all the links correctly, and DLL-manipulating code is even more error-prone than SLL-manipulating code. The great advantage of DLLs is their symmetry: we can access any DLL node's predecessor as easily as its successor, and we can access the DLL's last node as easily as its first node.

It is often convenient to view a DLL as a *backward SLL* superimposed on a *forward SLL*, as illustrated in Figure 4.10. If the DLL is headed by the pair of links (*first*, *last*), its forward SLL is headed by *first* and linked by successor links; and its backward SLL is headed by *last* and linked by predecessor links. The forward and backward SLLs share the same nodes.

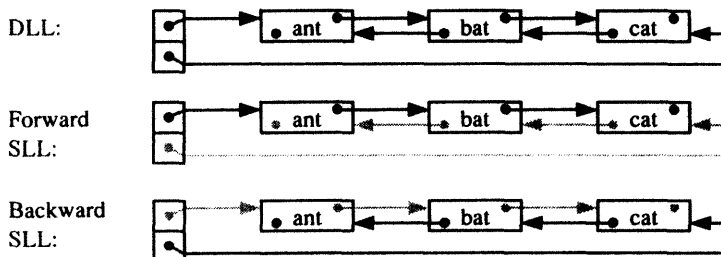


Figure 4.10 A DLL viewed as a backward SLL superimposed on a forward SLL.

4.1.3 Sorted linked lists

A (singly- or doubly-) linked list is *sorted* if the elements in its nodes are in ascending order, i.e., the element in each node of the linked list is less than or equal to the element in that node's successor. This definition is analogous to the definition of a sorted array in Section 3.1.2.

Sorted linked lists can be merged, and can be searched a little more efficiently than