



Figure 4.6 Doubly-linked lists.

The characteristic feature of an SLL is that each node contains a link to its successor only. This allows us to visit nodes *from first to last*. This was illustrated in different ways by Example 4.3 (accessing the second node from the first node) and Example 4.2 (traversing the whole SLL first-to-last).

The weakness of SLLs is that there is no easy way to visit nodes *from last to first*. This observation motivates us to consider an alternative data structure.

4.1.2 Doubly-linked lists

A *doubly-linked list* (or *DLL*) consists of a sequence of nodes, with the following properties:

- Each DLL node contains an element, together with a link to its predecessor (or null if it has no predecessor) and a link to its successor (or null if it has no successor).
- The DLL has a *header*, which contains a link to the DLL’s first node and a link to the DLL’s last node (both links being null if the DLL is empty).

Figure 4.6 shows various DLLs. (Compare with Figure 4.2.) The DLL (a) consists of three nodes, and its header contains links to the first and last (third) nodes. The DLL (c) has a single node, and its header contains two links to that single node. The DLL (d) is empty, and its header contains two null links.

In Java, we can represent DLL headers by objects of class `DLL`, shown in Program 4.7, and DLL nodes by objects of class `DLLNode`, shown in Program 4.8. The detailed structures of these objects are shown in Figure 4.9. Compared to an `SLLNode` object, each `DLLNode` object has an extra instance variable, `pred`, and both `succ` and `pred` are of type `DLLNode`.

EXAMPLE 4.4 *DLL construction*

The following Java code and diagrams illustrate construction of a DLL: