## Example scheme of linkedlist

pointer to the next node in the list.) See Figure 5-5.

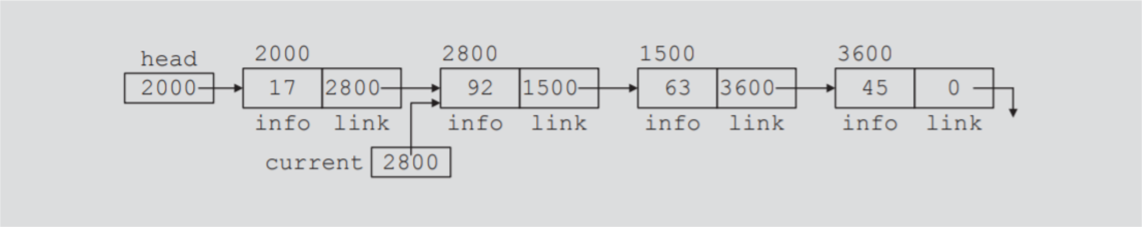
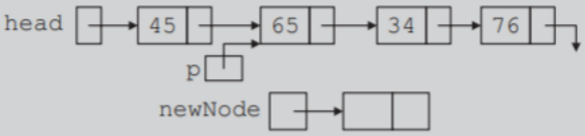
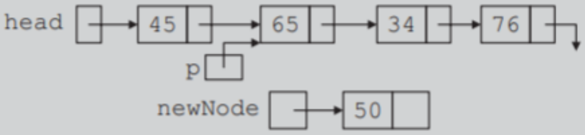
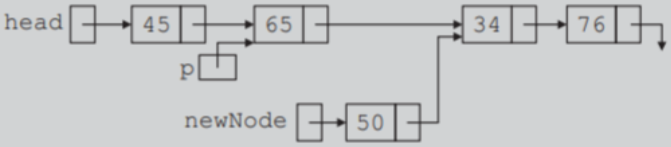
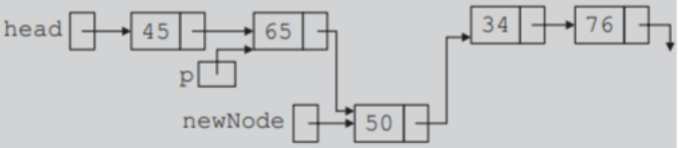


**FIGURE 5-5** List after the statement `current = current->link;` executes

## Insertion

**TABLE 5-3** Inserting a node in a linked list

| Statement | Effect |
|---|---|
| `newNode = new nodeType;` |  |
| `newNode->info = 50;` |  |
| `newNode->link = p->link;` |  |
| `p->link = newNode;` |  |

## Losing the linkedlist sequence

Figure 5-7 shows the resulting list after these statements execute.

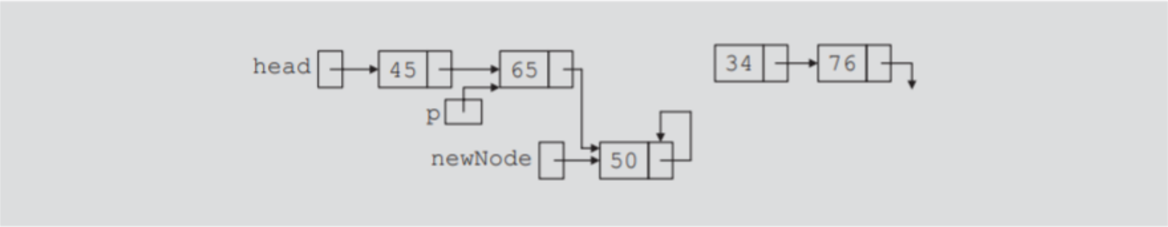
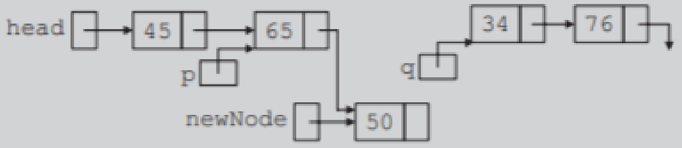
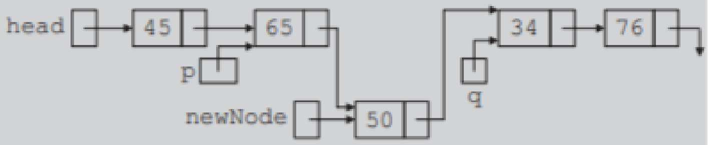


**FIGURE 5-7** List after the execution of the statement `p->link = newNode;` followed by the execution of the statement `newNode->link = p->link;`

**TABLE 5-4** Inserting a node in a linked list using two pointers

| Statement | Effect |
|---|---|
| `p->link = newNode;` |  |
| `newNode->link = q;` |  |

## DELETION

### DELETION

Consider the linked list shown in Figure 5-9.

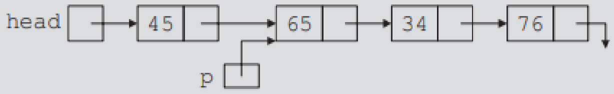


**FIGURE 5-9** Node to be deleted is with `info 34`

Suppose that the node with **`info 34`** is to be deleted from the list. The following statement removes the node from the list:

```
p->link = p->link->link;
```

Figure 5-10 shows the resulting list after the preceding statement executes.

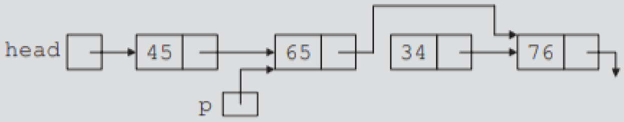
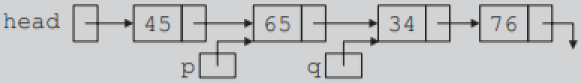
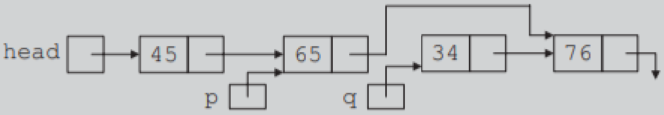
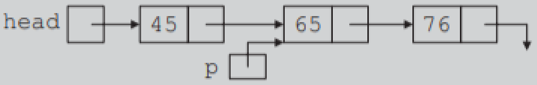


**FIGURE 5-10** List after the statement `p->link = p->link->link;` executes

**TABLE 5-5** Deleting a node from a linked list

| Statement | Effect |
|---|---|
| `q = p->link;` |  |
| `p->link = q->link;` |  |
| `delete q;` |  |

1. Mark the following statements as true or false.

   a. In a linked list, the order of the elements is determined by the order in which the nodes were created to store the elements.

   b. In a linked list, memory allocated for the nodes is sequential.

   c. A single linked list can be traversed in either direction.

   d. In a linked list, nodes are always inserted either at the beginning or the end because a linked link is not a random access data structure.

   e. The head pointer of a linked list cannot be used to traverse the list.

Consider the linked list shown in Figure 5-35. Assume that the nodes are in the usual **info-link** form. Use this list to answer Exercises 2 through 7. If necessary, declare additional variables. (Assume that **list**, **p**, **s**, **A**, and **B** are pointers of type **nodeType**.)

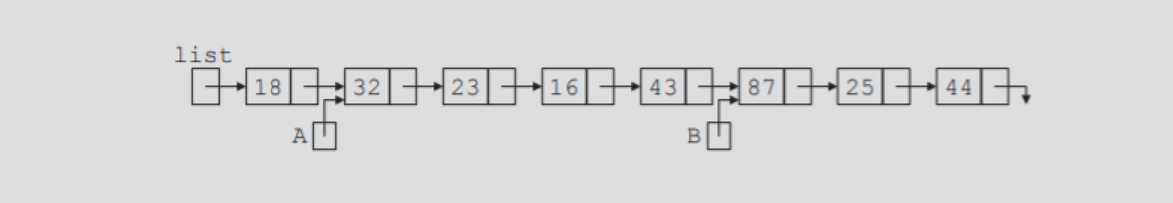


**FIGURE 5-35** Linked list for Exercises 2–7

2. What is the output of each of the following C++ statements?

   a. `cout << list->info;`

   b. `cout << A->info;`

   c. `cout << B->link->info;`

   d. `cout << list->link->link->info`

3. What is the value of each of the following relational expressions?

   a. `list->info >= 18`

   b. `list->link == A`

   c. `A->link->info == 16`

   d. `B->link == NULL`

   e. `list->info == 18`

4. Mark each of the following statements as valid or invalid. If a statement is invalid, explain why.

a. A = B;

b. list->link = A->link;

c. list->link->info = 45;

d. *list = B;

e. *A = *B;

f. B = A->link->info;

g. A->info = B->info;

h. list = B->link->link;

i. B = B->link->link->link;

i. B = B->link->link->link;

5. Write C++ statements to do the following:

a. Make A point to the node containing info 23.

b. Make list point to the node containing 16.

c. Make B point to the last node in the list.

d. Make list point to an empty list.

e. Set the value of the node containing 25 to 35.

f. Create and insert the node with info 10 after the node pointed to by A.

g. Delete the node with info 23. Also, deallocate the memory occupied by this node.