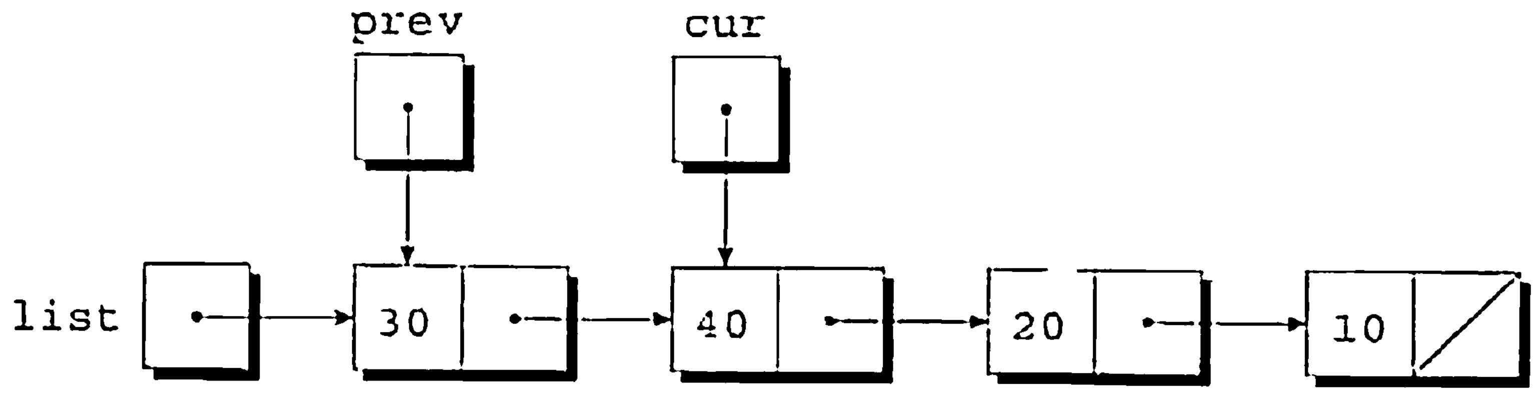
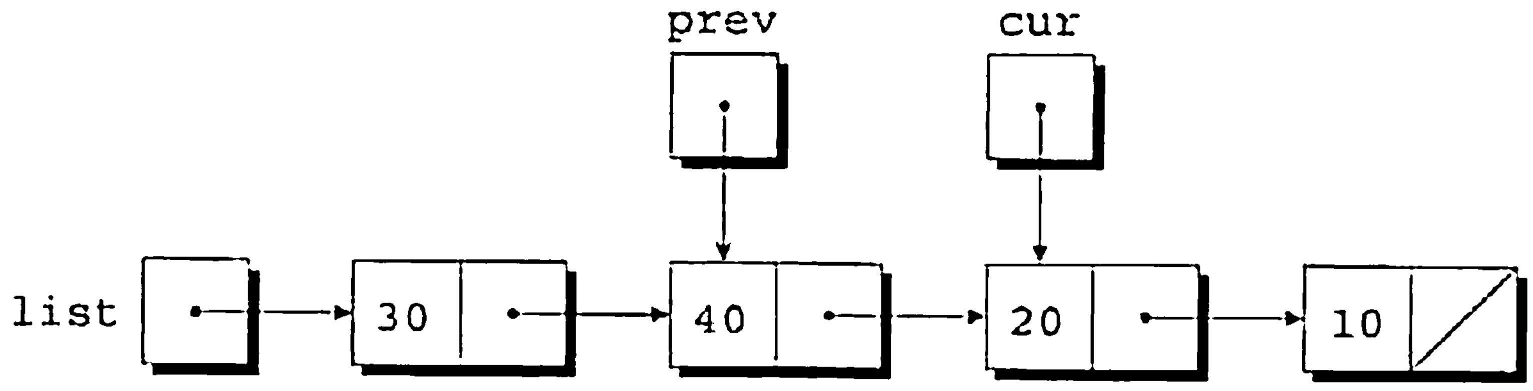


Again, the test cur != NULL && cur->value != n is true, so prev = cur, cur = cur->next is executed once more:

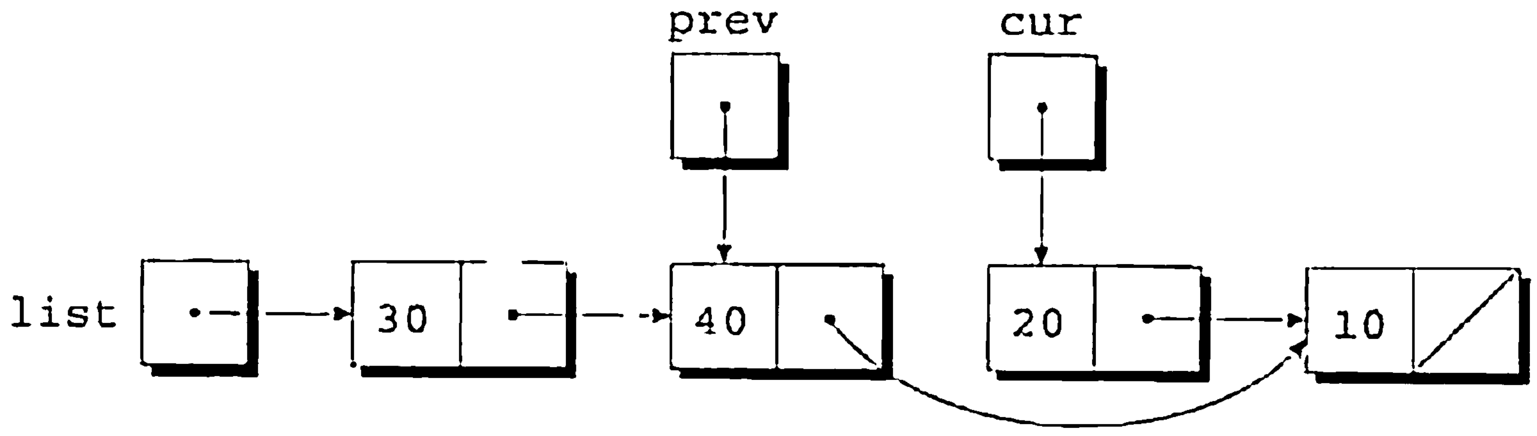


Since cur now points to the node containing 20, the condition cur->value != n is false and the loop terminates.

Next, we'll perform the bypass required by step 2. The statement

```
prev->next = cur->next;
```

makes the pointer in the previous node point to the node *after* the current node:



We're now ready for step 3, releasing the memory occupied by the current node:

```
free(cur);
```

The following function, delete_from_list, uses the strategy that we've just outlined. When given a list and an integer n, the function deletes the first node containing n. If no node contains n, delete_from_list does nothing. In either case, the function returns a pointer to the list.

```
struct node *delete_from_list(struct node *list, int n)
{
  struct node *cur, *prev;

  for (cur = list, prev = NULL;
       cur != NULL && cur->value != n;
       prev = cur, cur = cur->next)
    ;
```