Chapter 17:
Linked Lists:
Deleting a Node
Deleting a Node

- Used to remove a node from a linked list
- If list uses dynamic memory, then delete node from memory
- Requires two pointers:
  - one to locate the node to be deleted
  - one to point to the node before the node to be deleted
Traversing a Linked List

- Visit each node in a linked list: display contents, validate data, etc.

- Basic process:
  - set a pointer to the contents of the head pointer
  - while pointer is not a null pointer
    - process data
    - go to the next node by setting the pointer to the pointer field of the current node in the list
  - end while
Traversing a Linked List

nodePtr points to the node containing 5, then the node containing 13, then the node containing 19, then points to the null pointer, and the list traversal stops.
Deleting a Node: Example

Locating the node containing 13
Deleting a Node: Example

Adjusting pointer around the node to be deleted
Deleting a Node: Example

Linked list after deleting the node containing 13
Deleting a Node: Code

```cpp
void NumberList::deleteNode(double num)
{
    ListNode *nodePtr; // To traverse the list
    ListNode *previousNode; // To point to the previous node

    // If the list is empty, do nothing.
    if (!head)
        return;
```
Deleting a Node: Code

```c
// Determine if the first node is the one.
if (head->value == num)
{
    nodePtr = head->next;
    delete head;
    head = nodePtr;
}
else
{
    // Initialize nodePtr to head of list
    nodePtr = head;

    // Skip all nodes whose value member is
    // not equal to num.
    while (nodePtr != nullptr && nodePtr->value != num)
    {
        previousNode = nodePtr;
        nodePtr = nodePtr->next;
    }
}
```
Deleting a Node: Code

```c
// If nodePtr is not at the end of the list,
// link the previous node to the node after
// nodePtr, then delete nodePtr.
if (nodePtr)
{
    previousNode->next = nodePtr->next;
    delete nodePtr;
}
```
Program 17-4

1    // This program demonstrates the deleteNode member function.
2    #include <iostream>
3    #include "NumberList.h"
4    using namespace std;
5
6    int main()
7    {
8        // Define a NumberList object.
9        NumberList list;
10
11        // Build the list with some values.
12        list.appendNode(2.5);
13        list.appendNode(7.9);
14        list.appendNode(12.6);
15
16        // Display the list.
17        cout << "Here are the initial values:\n";
18        list.displayList();
19        cout << endl;
// Delete the middle node.
cout << "Now deleting the node in the middle.\n";
list.deleteNode(7.9);

// Display the list.
cout << "Here are the nodes left.\n";
list.displayList();
cout << endl;

// Delete the last node.
cout << "Now deleting the last node.\n";
list.deleteNode(12.6);

// Display the list.
cout << "Here are the nodes left.\n";
list.displayList();
cout << endl;
// Delete the only node left in the list.
cout << "Now deleting the only remaining node.\n";
list.deleteNode(2.5);

// Display the list.
cout << "Here are the nodes left.\n";
list.displayList();
return 0;
Program Output

Here are the initial values:
2.5
7.9
12.6

Now deleting the node in the middle.
Here are the nodes left.
2.5
12.6

Now deleting the last node.
Here are the nodes left.
2.5

Now deleting the only remaining node.
Here are the nodes left.
Destructor of LinkedList

- What must we destroy in our destructor?
- Dynamically allocated memory…

- When we insert a node we do what?
  - Create a new ListNode using dynamic memory allocation (newNode = new ListNode;)

© Pearson Education Limited 2015
Destroying a Linked List

- Must remove all nodes used in the list
- To do this, use list traversal to visit each node
- For each node,
  - Unlink the node from the list
  - If the list uses dynamic memory, then free the node’s memory
- Set the list head to NULL
Destroying Linked List: Code

```
NumberList::~NumberList()
{
    ListNode *nodePtr;    // To traverse the list
    ListNode *nextNode;  // To point to the next node

    // Position nodePtr at the head of the list.
    nodePtr = head;

    // While nodePtr is not at the end of the list...
    while (nodePtr != nullptr)
    {
        // Save a pointer to the next node.
        nextNode = nodePtr->next;

        // Delete the current node.
        delete nodePtr;

        // Position nodePtr at the next node.
        nodePtr = nextNode;
    }
}
```

Can we do this with only nodePtr?