DATA STRUCTURES AND ALGORITHMS
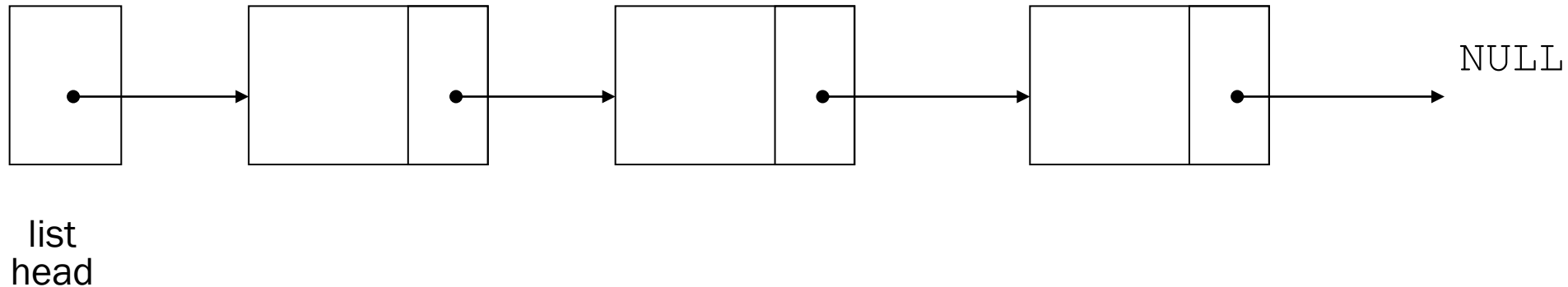
DR SAMABIA TEHSIN

BS (AI)

Introduction to the Linked List ADT
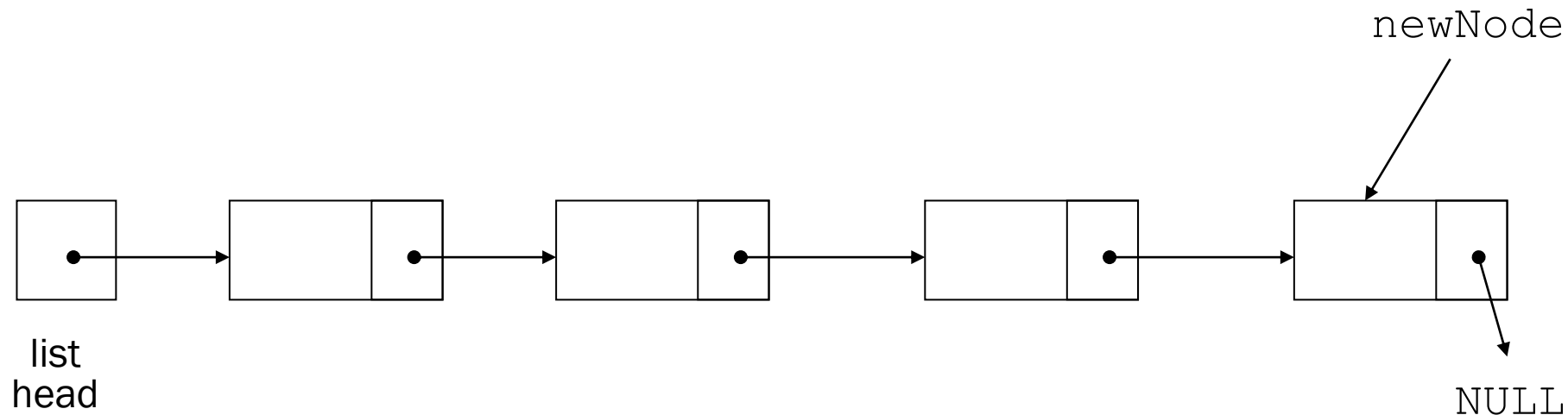
Linked list: set of data structures (nodes) that contain references to other data structures

list
head

Introduction to the Linked List ADT
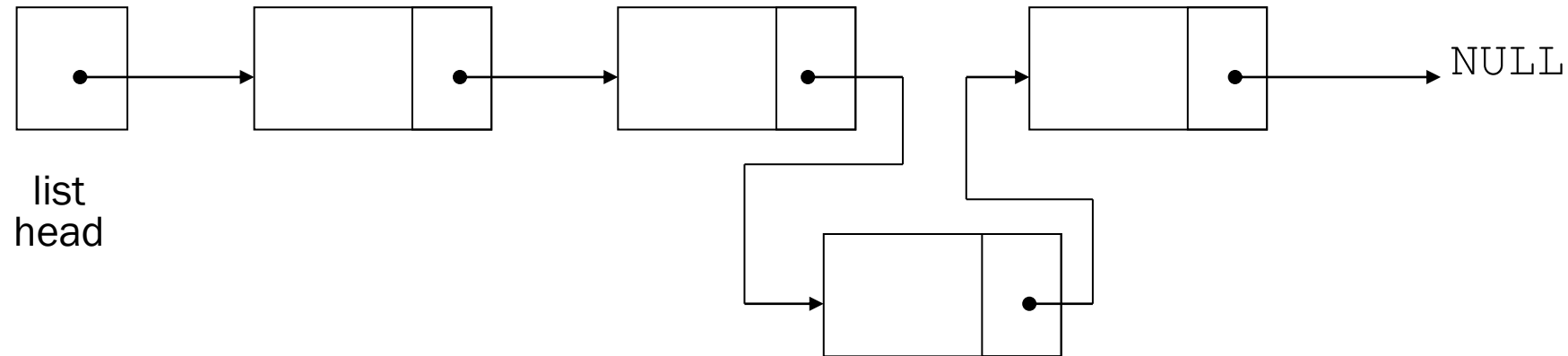
References may be addresses or array indices

Data structures can be added to or removed from the linked list during execution

Linked Lists vs. Arrays and Vectors

Linked lists can grow and shrink as needed, unlike arrays, which have a fixed size
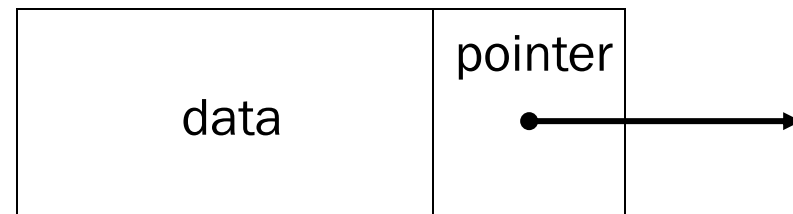
Linked lists can insert a node between other nodes easily

list
head

NULL

Node Organization

A node contains:

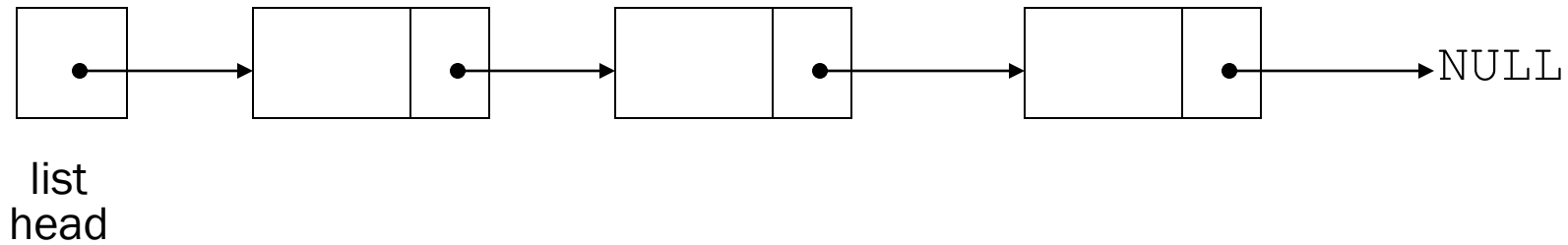◦ data: one or more data fields – may be organized as structure, object, etc.

◦ a pointer that can point to another node

Linked List Organization

Linked list contains 0 or more nodes:

list
head

Has a list head to point to first node

Last node points to `NULL`

Empty List

If a list currently contains 0 nodes, it is the <u>empty list</u>

In this case the list head points to `NULL`
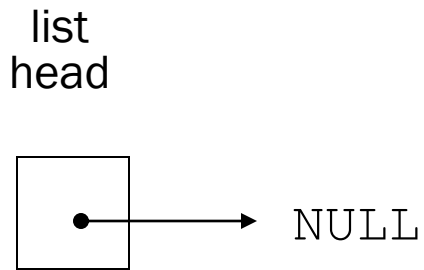
list
head

NULL

Declaring a Node

Declare a node:

```
struct ListNode
{
        int data;
        ListNode *next;
};
```

No memory is allocated at this time

Defining a Linked List

Define a pointer for the head of the list:

```
ListNode *head = NULL;
```

Head pointer initialized to `NULL` to indicate an empty list

```
head
```
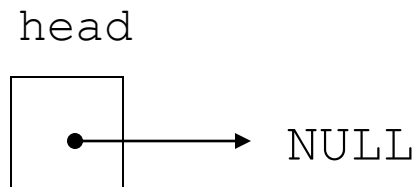
NULL Pointer

Is used to indicate end-of-list

Should always be tested for before using a pointer:
```
ListNode *p;
while (p != NULL) ...
```

Can also test the pointer itself:
```
while (!p) ... // same meaning
                       // as above
```

Linked List Operations

Basic operations:

◦ append a node to the end of the list

◦ insert a node within the list

◦ traverse the linked list

◦ delete a node

◦ delete/destroy the list

Contents of `NumberList.h`

```cpp
1   // Specification file for the NumberList class
2   #ifndef NUMBERLIST_H
3   #define NUMBERLIST_H
4
5   class NumberList
6   {
7   private:
8       // Declare a structure for the list
9       struct ListNode
10      {
11          double value;               // The value in this node
12          struct ListNode *next;   // To point to the next node
13      };
14
15      ListNode *head;                 // List head pointer
16
```

Contents of **NumberList.h** (Continued)

```
17  public:
18      // Constructor
19      NumberList()
20          { head = NULL; }
21
22      // Destructor
23      ~NumberList();
24
25      // Linked list operations
26      void appendNode(double);
27      void insertNode(double);
28      void deleteNode(double);
29      void displayList() const;
30  };
31  #endif
```

Create a New Node

Allocate memory for the new node:

```
newNode = new ListNode;
```

Initialize the contents of the node:

```
newNode->value = num;
```

Set the pointer field to `NULL`:

```
newNode->next = NULL;
```
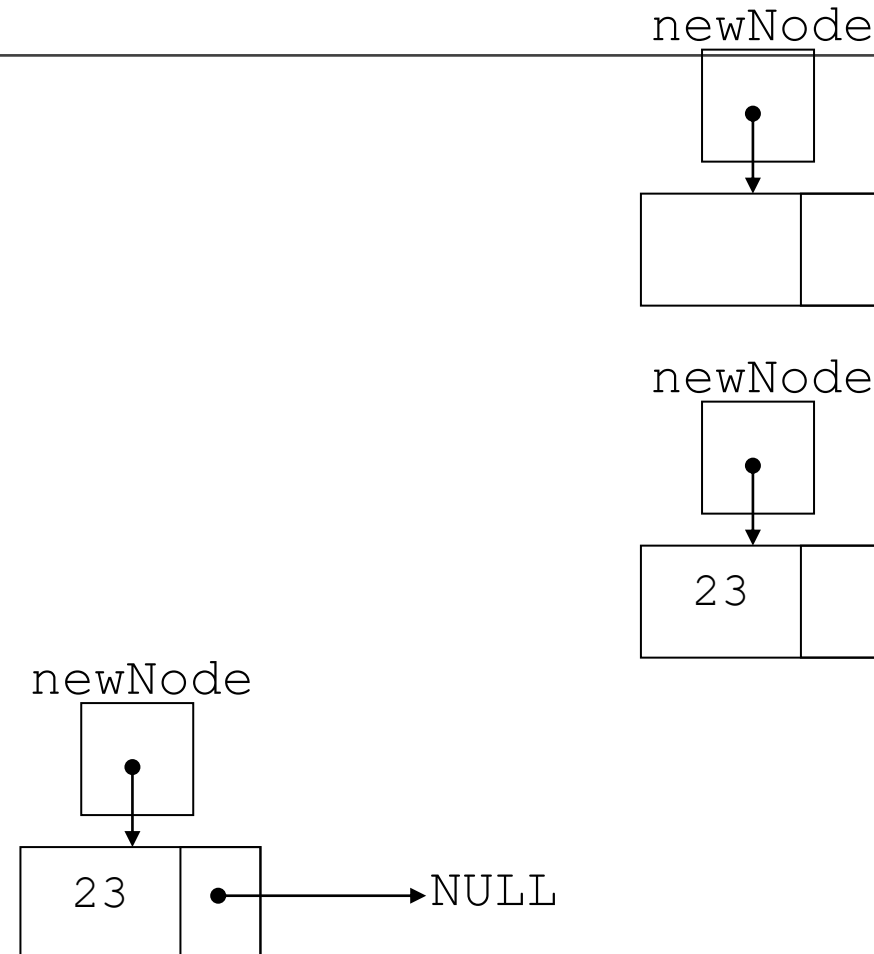
Appending a Node

Add a node to the end of the list

Basic process:
- Create the new node (as already described)
- Add node to the end of the list:
  - If list is empty, set head pointer to this node
  - Else,
    - traverse the list to the end
    - set pointer of last node to point to new node

Appending a Node
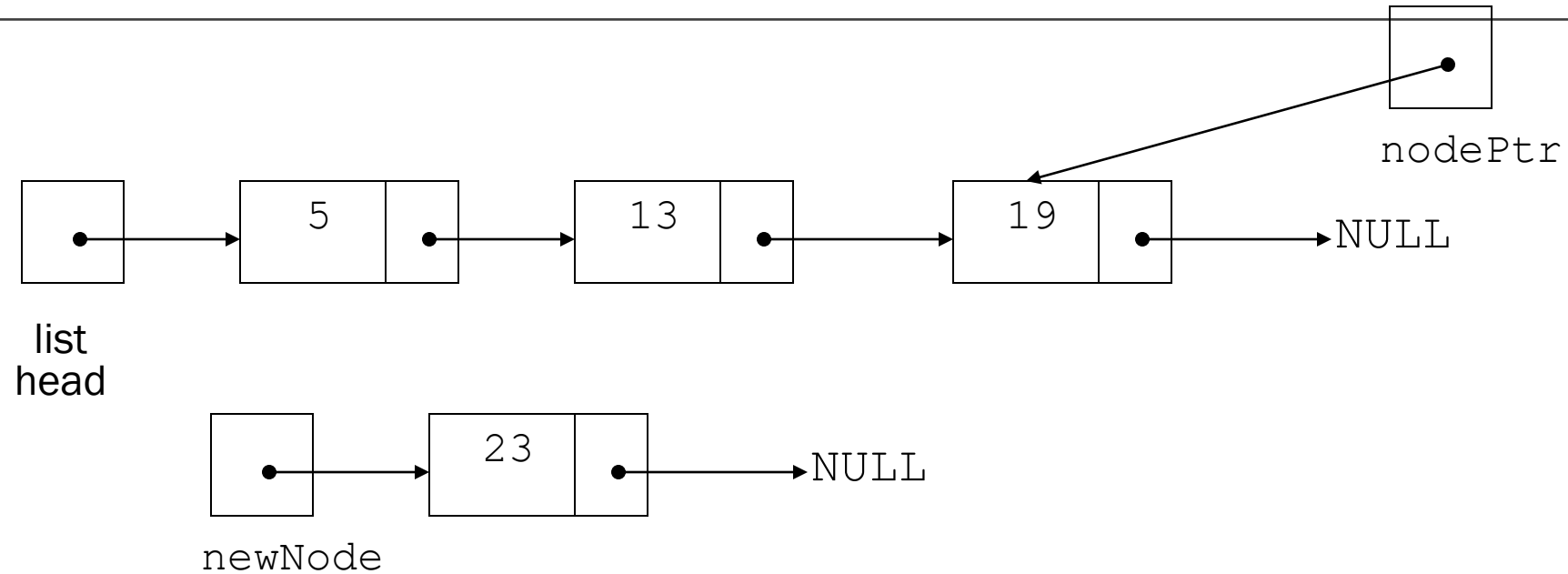
New node created, end of list located

Appending a Node

list head
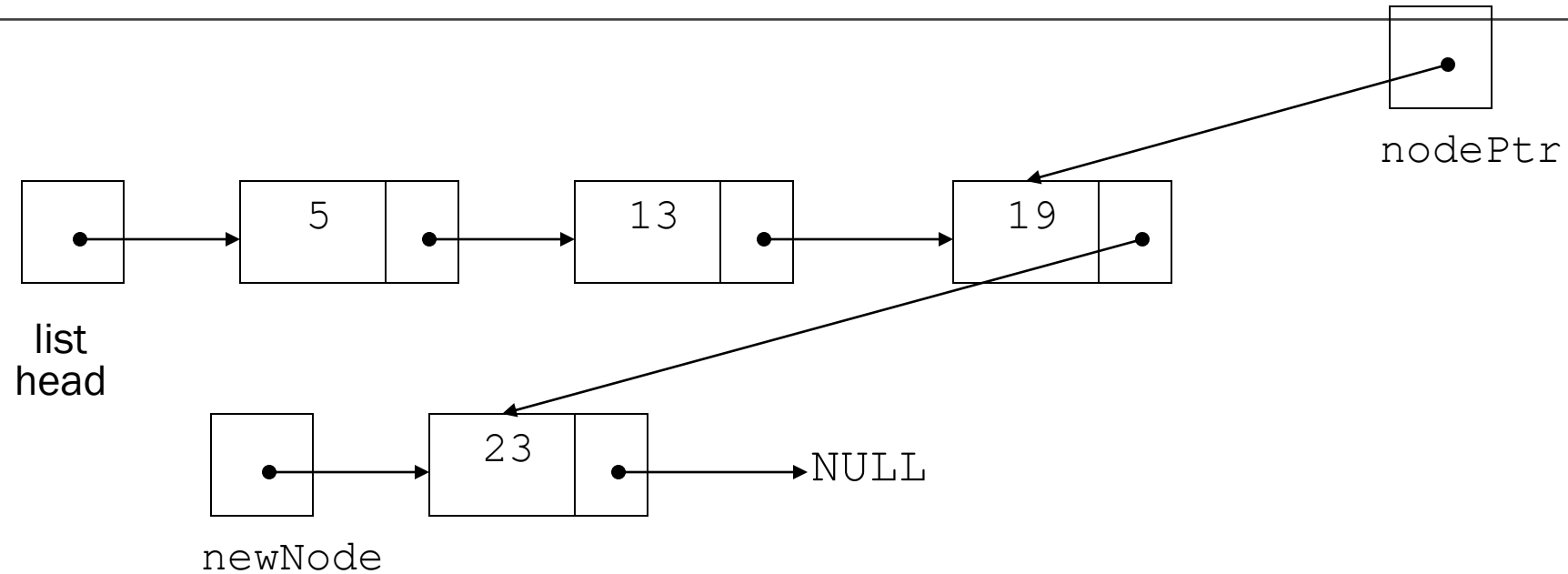
nodePtr

newNode

New node added to end of list

C++ code for Appending a Node

```cpp
11  void NumberList::appendNode(double num)
12  {
13      ListNode *newNode;   // To point to a new node
14      ListNode *nodePtr;   // To move through the list
15
16      // Allocate a new node and store num there.
17      newNode = new ListNode;
18      newNode->value = num;
19      newNode->next = NULL;
20
21      // If there are no nodes in the list
22      // make newNode the first node.
23      if (!head)
```

C++ code for Appending a Node (Continued)

```cpp
24              head = newNode;
25      else  // Otherwise, insert newNode at end.
26      {
27          // Initialize nodePtr to head of list.
28          nodePtr = head;
29
30          // Find the last node in the list.
31          while (nodePtr->next)
32              nodePtr = nodePtr->next;
33
34          // Insert newNode as the last node.
35          nodePtr->next = newNode;
36      }
37  }
```

Program 17-1

```
 1    // This program demonstrates a simple append
 2    // operation on a linked list.
 3    #include <iostream>
 4    #include "NumberList.h"
 5    using namespace std;
 6
 7    int main()
 8    {
 9        // Define a NumberList object.
10        NumberList list;
11
12        // Append some values to the list.
13        list.appendNode(2.5);
14        list.appendNode(7.9);
15        list.appendNode(12.6);
16        return 0;
17    }
```

(This program displays no output.)

Inserting a Node into a Linked List

Used to maintain a linked list in order

Requires two pointers to traverse the list:
- pointer to locate the node with data value greater than that of node to be inserted
- pointer to 'trail behind' one node, to point to node before point of insertion

New node is inserted between the nodes pointed at by these pointers

Inserting a Node into a Linked List

previousNode   nodePtr

5   13   19   NULL

list
head

17   NULL
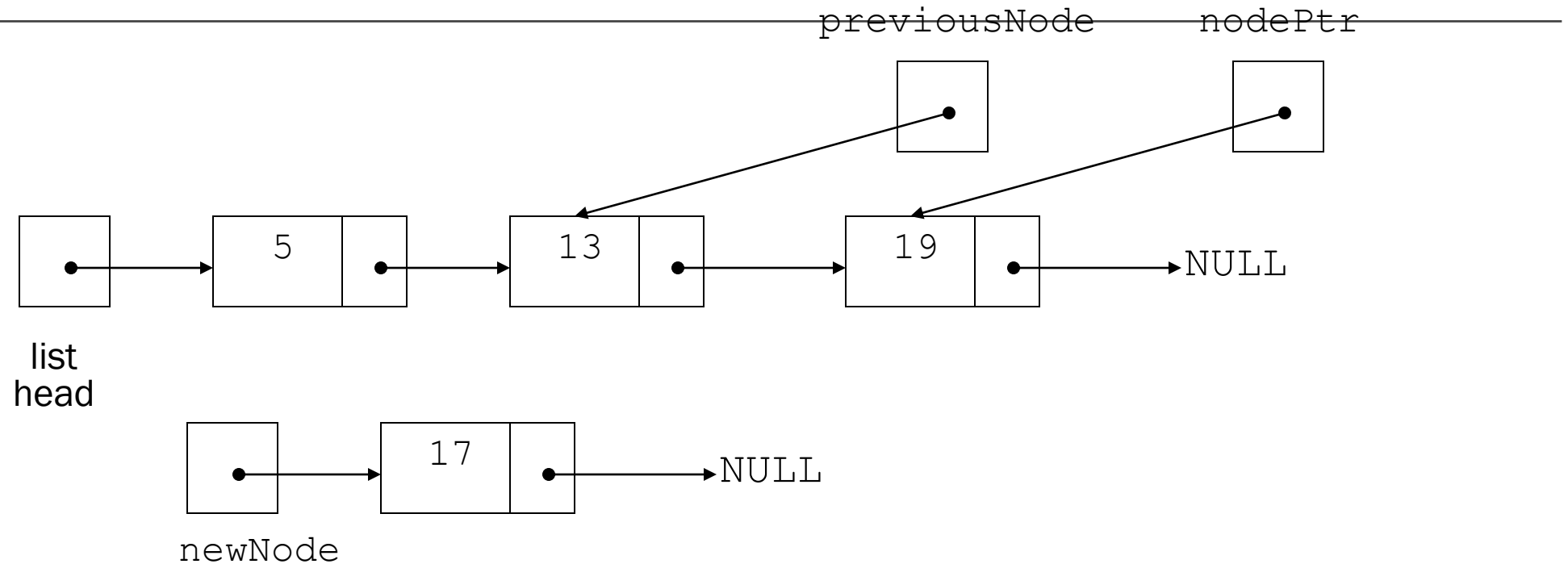
newNode

New node created, correct position located

Inserting a Node into a Linked List

previousNode   nodePtr

5 → 13 → 19 → NULL

list
head

17
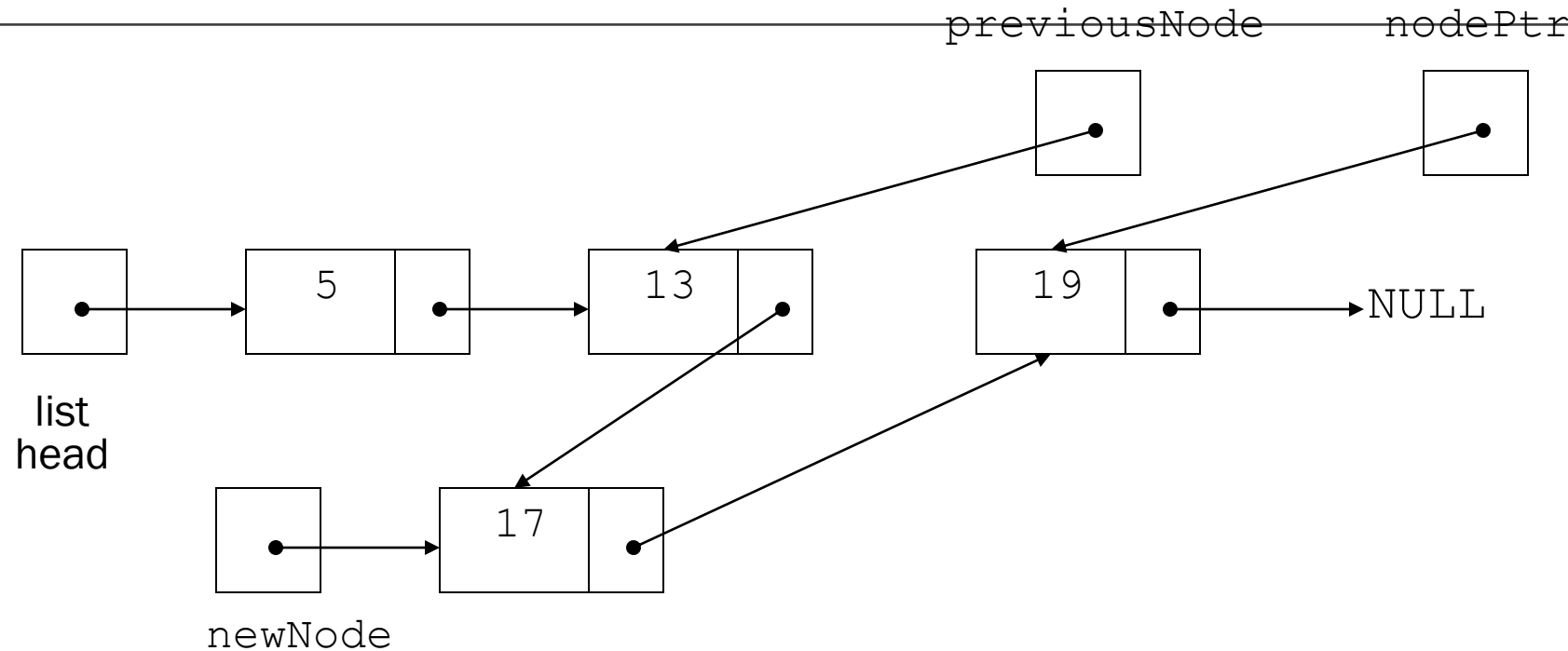
newNode

New node inserted in order in the linked list

```cpp
69   void NumberList::insertNode(double num)
70   {
71      ListNode *newNode;                    // A new node
72      ListNode *nodePtr;                     // To traverse the list
73      ListNode *previousNode = NULL; // The previous node
74
75      // Allocate a new node and store num there.
76      newNode = new ListNode;
77      newNode->value = num;
78
79      // If there are no nodes in the list
80      // make newNode the first node
81      if (!head)
82      {
83         head = newNode;
84         newNode->next = NULL;
85      }
86      else  // Otherwise, insert newNode
87      {
88         // Position nodePtr at the head of list.
89         nodePtr = head;
90
```

```cpp
91          // Initialize previousNode to NULL.
92          previousNode = NULL;
93
94          // Skip all nodes whose value is less than num.
95          while (nodePtr != NULL && nodePtr->value < num)
96          {
97              previousNode = nodePtr;
98              nodePtr = nodePtr->next;
99          }
100
101         // If the new node is to be the 1st in the list,
102         // insert it before all other nodes.
103         if (previousNode == NULL)
104         {
```

```
105            head = newNode;
106            newNode->next = nodePtr;
107        }
108    else  // Otherwise insert after the previous node.
109        {
110            previousNode->next = newNode;
111            newNode->next = nodePtr;
112        }
113    }
114 }
```

Program 17-3

```
1   // This program demonstrates the insertNode 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      // Insert a node in the middle of the list.
17      list.insertNode(10.5);
18
19      // Dispay the list
20      list.displayList();
21      return 0;
22  }
```

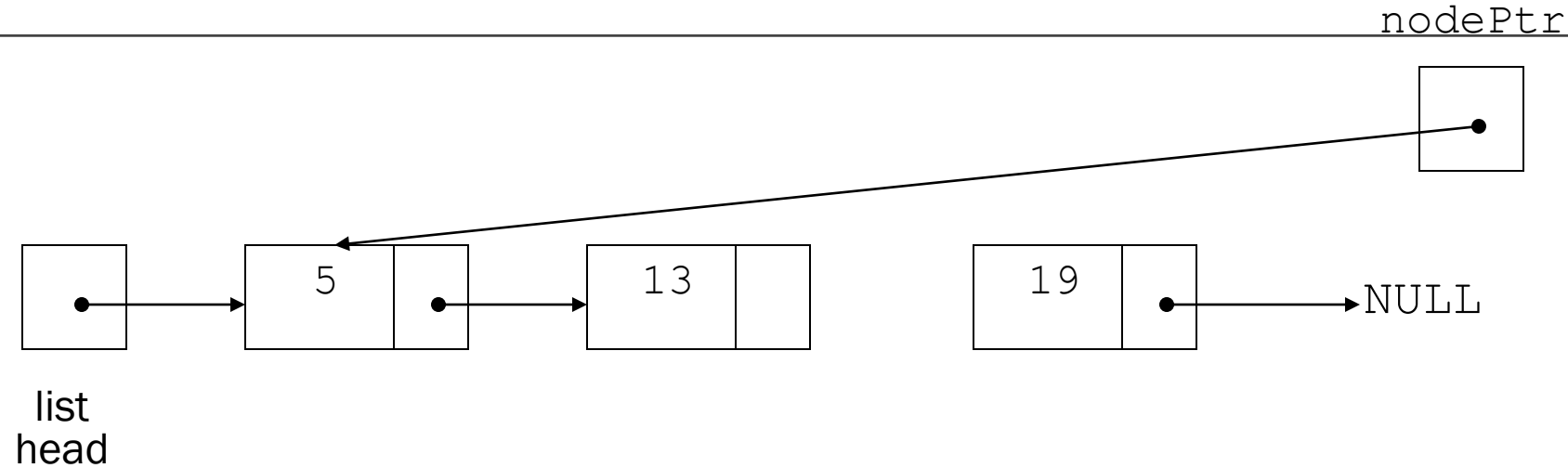**Program Output**

```
2.5
7.9
10.5
12.6
```

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 `NULL`
  - 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 `NULL`, and the list traversal stops
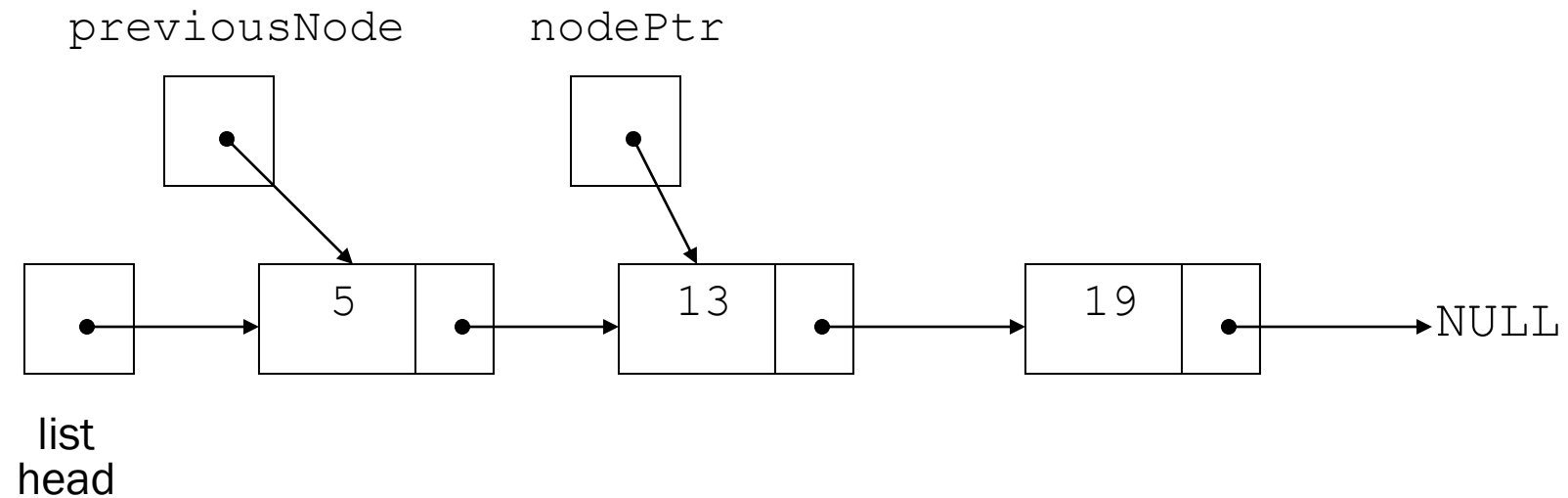
Deleting a Node

Used to remove a node from a linked list

If list uses dynamic memory, then delete node from memory
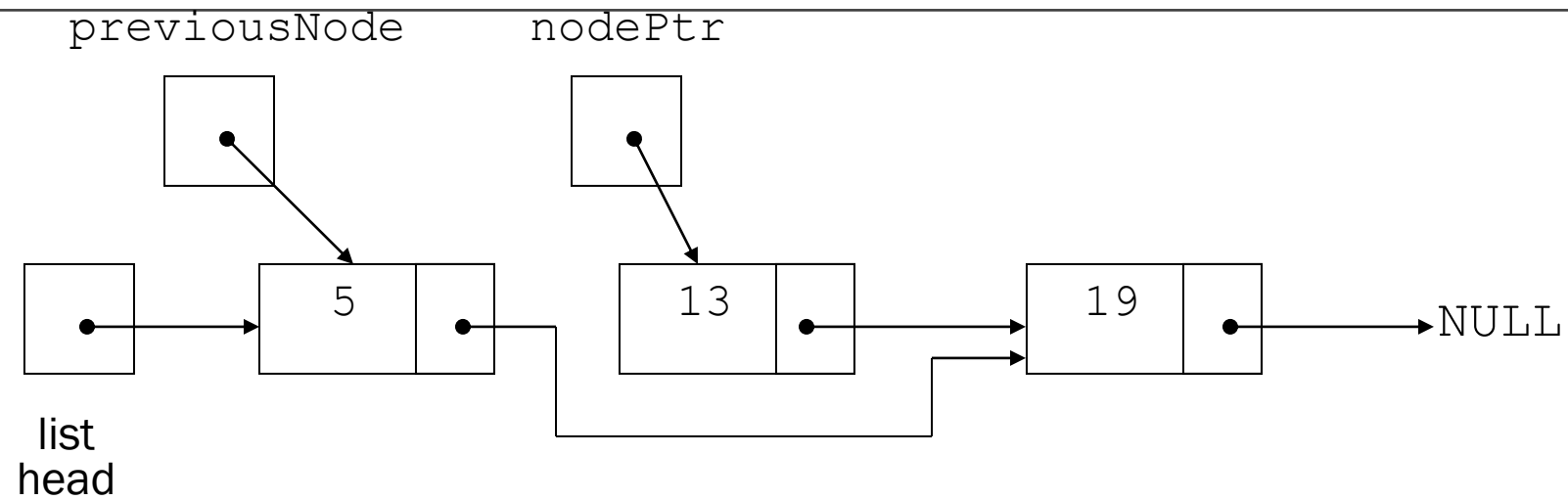
Requires two pointers: one to locate the node to be deleted, one to point to the node before the node to be deleted

Deleting a Node

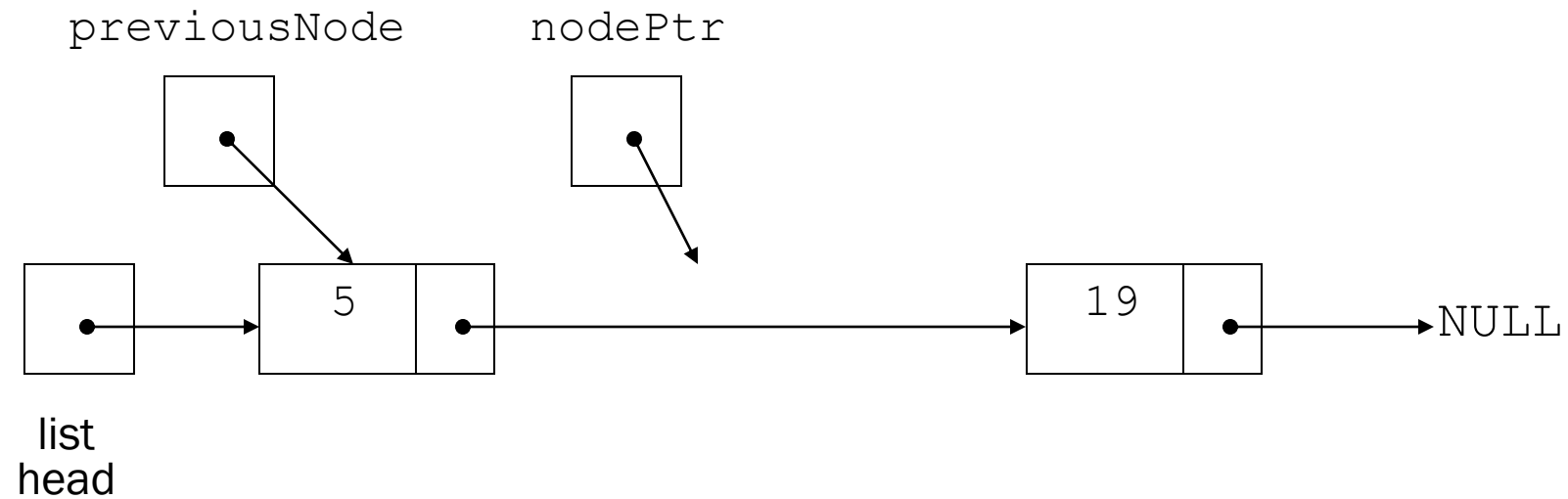previousNode   nodePtr

list
head

Locating the node containing 13

Deleting a Node

Adjusting pointer around the node to be deleted

Deleting a Node

previousNode   nodePtr

list
head

Linked list after deleting the node containing `13`

```
122  void NumberList::deleteNode(double num)
123  {
124      ListNode *nodePtr;          // To traverse the list
125      ListNode *previousNode;   // To point to the previous node
126
127      // If the list is empty, do nothing.
128      if (!head)
129          return;
130
131      // Determine if the first node is the one.
132      if (head->value == num)
133      {
134          nodePtr = head->next;
135          delete head;
136          head = nodePtr;
137      }
138      else
139      {
```

```
139      {
140          // Initialize nodePtr to head of list
141          nodePtr = head;
142
143          // Skip all nodes whose value member is
144          // not equal to num.
145          while (nodePtr != NULL && nodePtr->value != num)
146          {
147              previousNode = nodePtr;
148              nodePtr = nodePtr->next;
149          }
150
151          // If nodePtr is not at the end of the list,
152          // link the previous node to the node after
153          // nodePtr, then delete nodePtr.
154          if (nodePtr)
155          {
156              previousNode->next = nodePtr->next;
157              delete nodePtr;
158          }
159      }
160 }
```

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;
20
```

```
21        // Delete the middle node.
22        cout << "Now deleting the node in the middle.\n";
23        list.deleteNode(7.9);
24
25        // Display the list.
26        cout << "Here are the nodes left.\n";
27        list.displayList();
28        cout << endl;
29
30        // Delete the last node.
31        cout << "Now deleting the last node.\n";
32        list.deleteNode(12.6);
33
34        // Display the list.
35        cout << "Here are the nodes left.\n";
36        list.displayList();
37        cout << endl;
38
```

```
39        // Delete the only node left in the list.
40        cout << "Now deleting the only remaining node.\n";
41        list.deleteNode(2.5);
42
43        // Display the list.
44        cout << "Here are the nodes left.\n";
45        list.displayList();
46        return 0;
47   }
```

Program 17-4 *(continued)*

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.
```

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`

```cpp
167   NumberList::~NumberList()
168   {
169       ListNode *nodePtr;    // To traverse the list
170       ListNode *nextNode;   // To point to the next node
171
172       // Position nodePtr at the head of the list.
173       nodePtr = head;
174
175       // While nodePtr is not at the end of the list...
176       while (nodePtr != NULL)
177       {
178           // Save a pointer to the next node.
179           nextNode = nodePtr->next;
180
181           // Delete the current node.
182           delete nodePtr;
183
184           // Position nodePtr at the next node.
185           nodePtr = nextNode;
186       }
187   }
```

Credits and Acknowledgements

Chapter 17 Linked Lists Copyright Pearson Education