Chapter 18:

Linked Lists

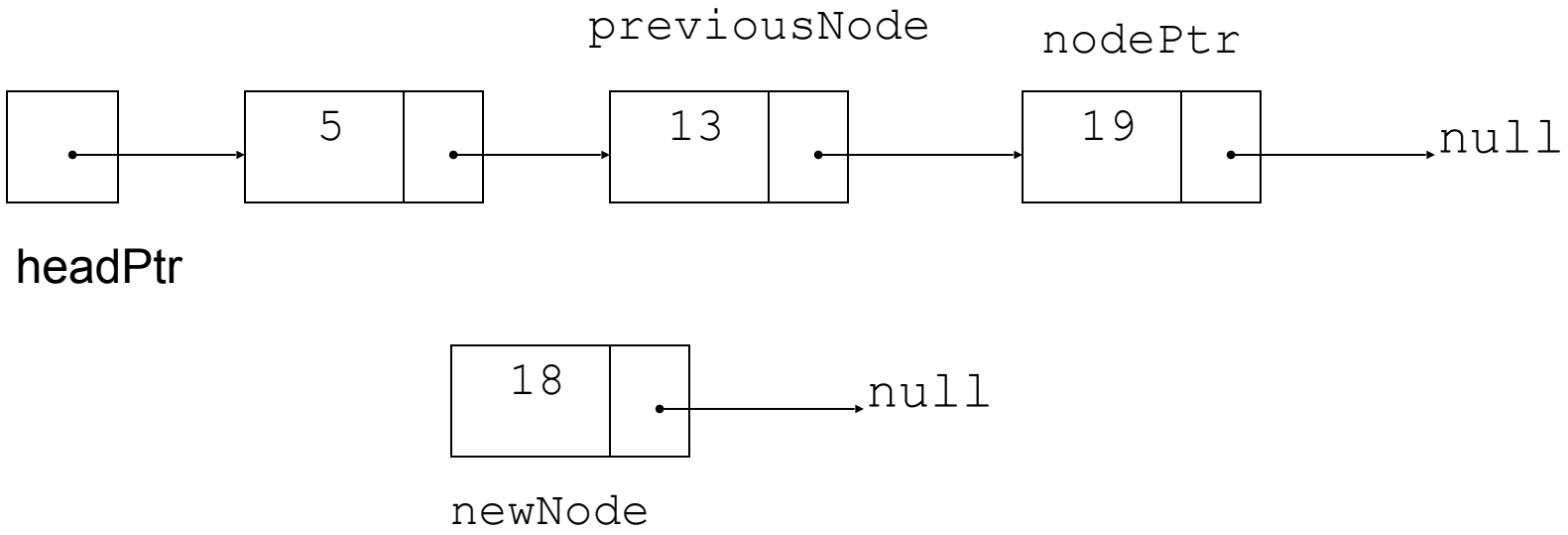
18.2

Linked List Operations

Inserting a Node into a Linked List

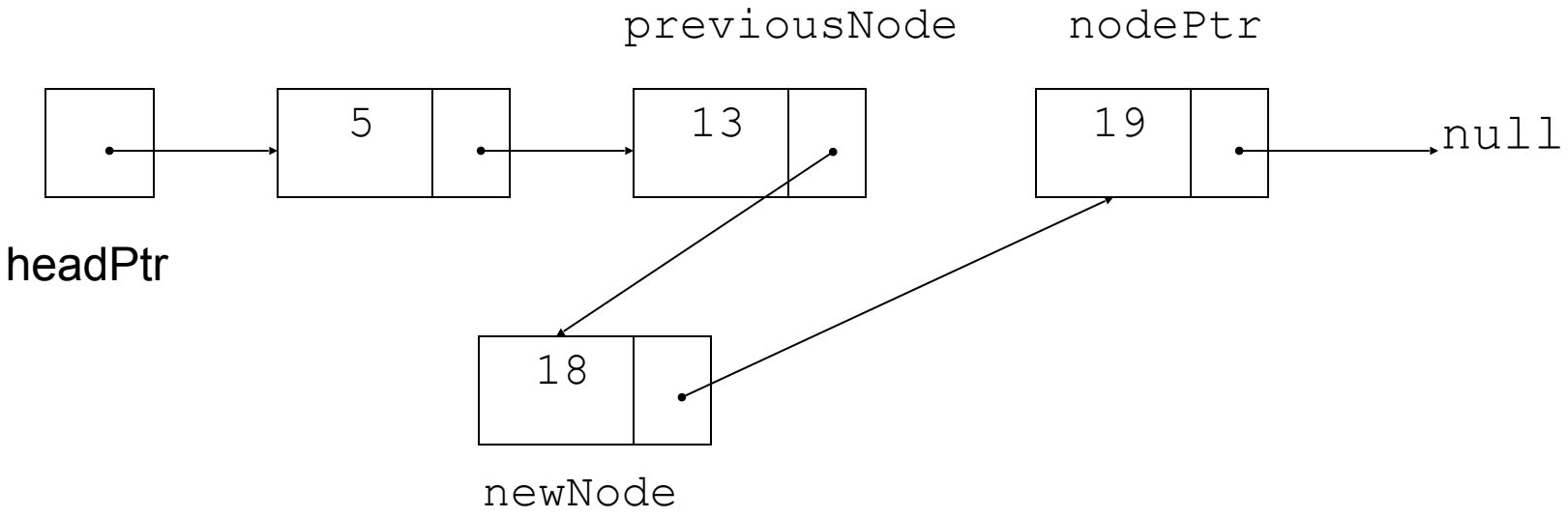
- Used to maintain a linked list in order
- Requires two pointers to traverse the list:
  - pointer to locate the node with data value greater than that of node to be inserted
  - pointer to 'trail behind' one node, to point to node before point of insertion
- New node is inserted between the nodes pointed at by these pointers

Inserting a Node into a Linked List



New node created, correct position located

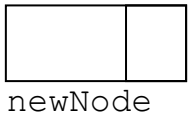
Inserting a Node into a Linked List

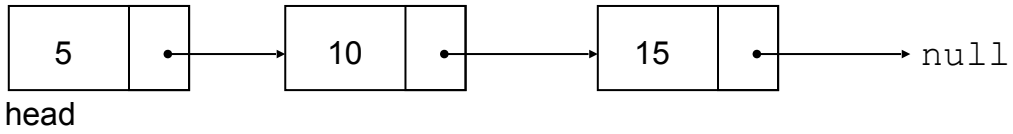


New node inserted in order in the linked list

```
69  void NumberList::insertNode(double num)
70  {
71       ListNode *newNode;                 // A new node
72       ListNode *nodePtr;                 // To traverse the list
73       ListNode *previousNode = nullptr; // 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 = nullptr;
85       }
86       else // Otherwise, insert newNode
87       {
88            // Position nodePtr at the head of list.
89            nodePtr = head;
90
```

```
91            // Initialize previousNode to nullptr.
92            previousNode = nullptr;
93
94            // Skip all nodes whose value is less than num.
95            while (nodePtr != nullptr && 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 == nullptr)
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 18-3

```
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      // Display the list
20      list.displayList();
21      return 0;
22 }
```

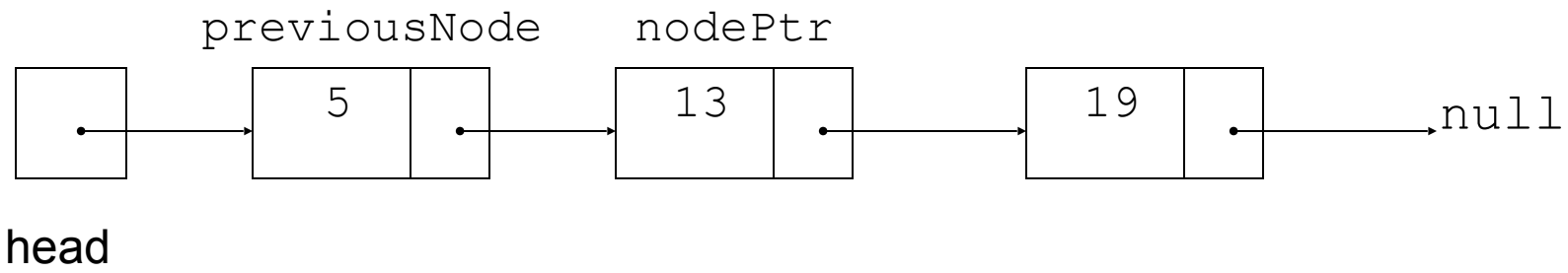
Program Output

```
2.5
7.9
10.5
12.6
```

Deleting a Node

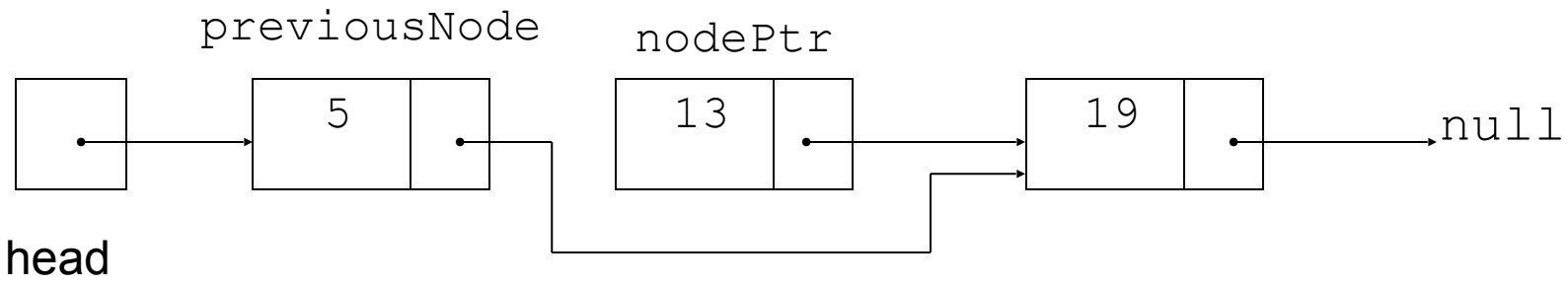
- Used to remove a node from a linked list
- If list uses dynamic memory, then delete node from memory
- Requires two pointers:
  - one to locate the node to be deleted
  - another that points to the previous node

Deleting a Node



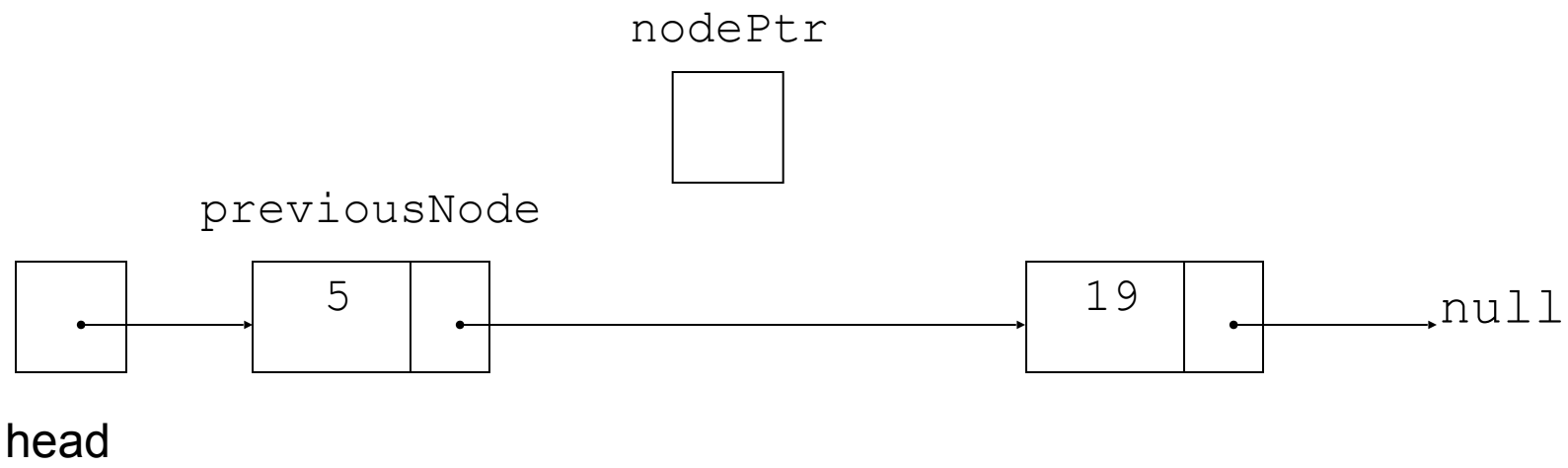
Locating the node containing `13`

Deleting a Node



Adjusting pointer around the node to be deleted

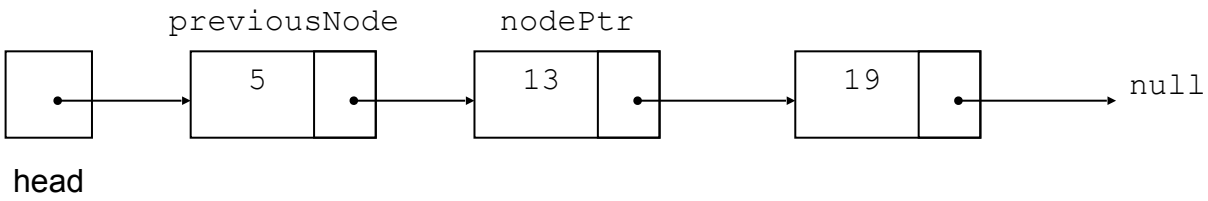
Deleting a Node



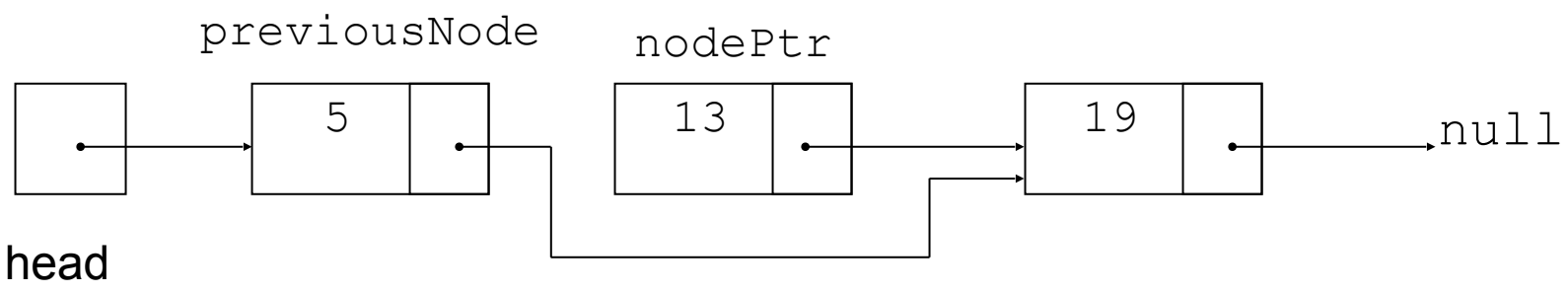
Linked list after deleting the node containing `13`

```
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         {
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 != nullptr && 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 18-4

```
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 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 `nullptr`

```
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 != nullptr)
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  }
```

18.3

A Linked List Template

A Linked List Template

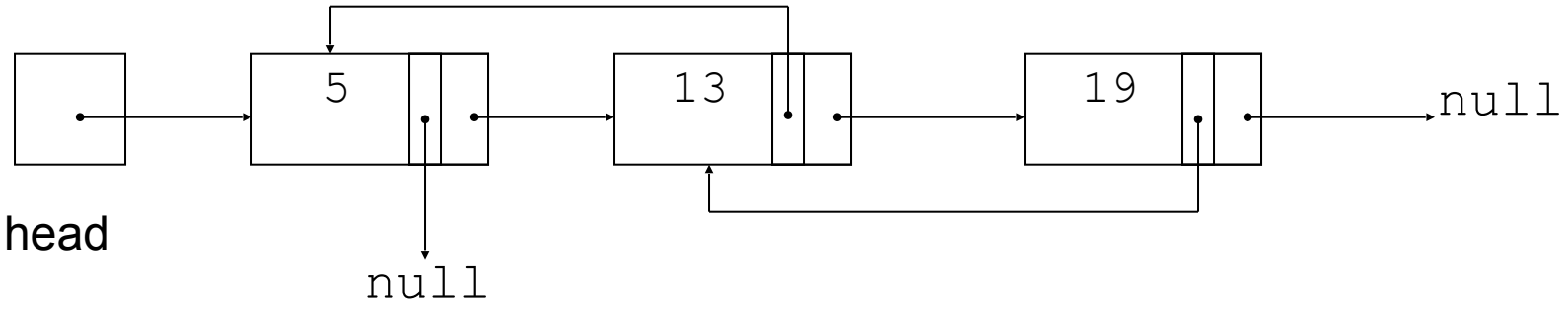
- When declaring a linked list, must specify the type of data to be held in each node
- Using templates, can declare a linked list that can hold data type determined at list definition time
- See `LinkedList.h` (versions 1 and 2) and Program 18-5

18.4

Variations of the Linked List

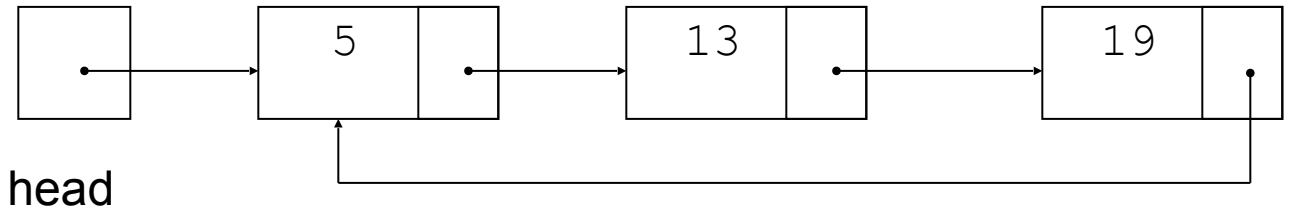
Variations of the Linked List

- Other linked list organizations:
  - doubly-linked list: each node contains two pointers: one to the next node in the list, one to the previous node in the list



Variations of the Linked List

- Other linked list organizations:
  - circular linked list: the last node in the list points back to the first node in the list, not to the null pointer



18.5

The STL `list` and `forward_list` Containers

The STL `list` Container

- Template for a doubly linked list
- Member functions for
  - locating beginning, end of list: `front`, `back`, `end`
  - adding elements to the list: `insert`, `merge`, `push_back`, `push_front`
  - removing elements from the list: `erase`, `pop_back`, `pop_front`, `unique`
- See Table 18-1 for a list of constructors
- See Table 18-2 for a list of member functions

The STL `forward_list` Container

- Template for a singly linked list
- You can only step forward in a `forward_list`.
- A `forward_list` uses slightly less memory than a `list`, and takes slightly less time for inserting and removing nodes.
- Provides most, but not all, of the same member functions as the `list` container

Thank You

Please let me know if you have any questions.