

# Recursive Linked Lists

Lecture 28  
Section 20.5

Robb T. Koether

Hampden-Sydney College

Mon, Apr 9, 2018

## 1 Recursive Linked Lists

## 2 Recursive Member Functions

- The `locateNode()` Function
- The `insert()` Function
- The `remove()` Function

## 3 Assignment

## 1 Recursive Linked Lists

## 2 Recursive Member Functions

- The `locateNode()` Function
- The `insert()` Function
- The `remove()` Function

## 3 Assignment

# Recursive Linked Lists

- A linked list is a naturally recursive structure.
- The “linked list” is a pointer to a node. (Ignore the `m_size` data member.)
- Furthermore, each node contains a pointer to a node.
- Therefore, each node contains a “linked list.”
- However, the sublists do not have an `m_size` data member, so they are *not exactly* `LinkedList` objects.

# Outline

## 1 Recursive Linked Lists

## 2 Recursive Member Functions

- The `locateNode()` Function
- The `insert()` Function
- The `remove()` Function

## 3 Assignment

# Two-Part Implementation of Recursive Member Functions

## Recursive Member Functions

```
Type Class::func(parameters)    // Non-recursive func()
{
    {Do some special things...}
    {Call the second function,
    {passing the head of the list}
    func(m_head, parameters);    // Recursive func()
    {Do more special things...}
    return;
}
```

- We implement the recursive functions in two parts.
- The first function is public and nonrecursive.

# Implementation of Recursive Member Functions

## Example (A Typical Recursive Function)

```
ret-type Class::func(LinkedListNode* node, parameters)
{
    if (condition)
    {
        // Handle the recursive case
        return func(node->next, parameters);
    }
    else
    {
        // Handle the nonrecursive case
    }
    return;
}
```

# Outline

1 Recursive Linked Lists

2 Recursive Member Functions

- The `locateNode()` Function
- The `insert()` Function
- The `remove()` Function

3 Assignment



# The Nonrecursive locateNode () Function

## Example (The Nonrecursive locateNode () Function)

```
LinkedListNode<T>* locateNode(int pos) const
{
    assert(pos >= 0 && pos < m_size);
    return locateNode(m_head, pos);
}
```

# The Recursive locateNode () Function

## Example (The Recursive locateNode () Function)

```
LinkedListNode<T>* locateNode(LinkedListNode<T>* node,  
int pos) const  
{  
    if (pos == 0)  
        return node;  
    else  
        return locateNode(node->m_next, pos - 1);  
}
```

# Outline

## 1 Recursive Linked Lists

## 2 Recursive Member Functions

- The `locateNode()` Function
- **The `insert()` Function**
- The `remove()` Function

## 3 Assignment

# Example

## Example (The `insert()` Function)

- There are two things that the nonrecursive `insert()` function should do only once.
  - Check that `pos` is valid.
  - Increment `m_size`.
- Then it makes a call to the recursive `insert()` function, passing it the head pointer.

# The Nonrecursive `insert()` Function

## Example (The Nonrecursive `insert()` Function)

```
void insert(int pos, const T& value)
{
    {Do something special}
    assert(pos >= 0 && pos <= m_size);
    {Call the recursive function}
    insert(m_head, pos, value);
    {Do something special}
    m_size++;
    return;
}
```

# The Recursive `insert()` Function

## Example (The Recursive `insert()` Function)

- The recursive `insert()` function must distinguish two cases.
  - Recursive case - The insertion takes place at a later node.
  - Non-recursive case - The insertion takes place at the current node, which is the “head” of the (sub)list.

# The Recursive Case

## Example (The Recursive Case)

- The recursive case
  - Is distinguished by the condition that  $pos > 0$  (action at a later node).
  - Passes a pointer to the next node.
  - Decrements the value of  $pos$ .

# The Recursive Case

## Example (The Recursive Case)

```
if (pos > 0)
    insert(node->m_next, pos - 1, value);
    :
```



# The Non-Recursive Case

## Example (The Non-Recursive Case)

- The non-recursive case
  - Is distinguished by the condition  $pos == 0$  (action at this node).
  - Inserts the new node at the head of the (sub)list.
- Because `node` is modified, it must be passed by reference.

# The Non-Recursive Case

## Example (The Non-Recursive Case)

```
if (pos == 0)
{
    LinkedListNode<T>* new_node
        = new LinkedListNode<T>(value);
    new_node->m_next = node;
    node = new_node;
}
```

# The Recursive `insert()` Function

## Example (Recursive `insert()` Function)

```
void insert(LinkedListNode<T>*& node, int pos,
            const T& value)
{
// Recursive case
if (pos > 1)
    insert(node->m_next, pos - 1, value);
// Non-recursive case
else
{
    LinkedListNode<T>* new_node =
        new LinkedListNode<T>(value);
    new_node->m_next = node;
    node = new_node;
}
return;
}
```

# Outline

1 Recursive Linked Lists

2 Recursive Member Functions

- The `locateNode()` Function
- The `insert()` Function
- The `remove()` Function

3 Assignment

# The Recursive `remove()` Function

- Write a recursive implementation of the `remove()` function.

- 1 Recursive Linked Lists
- 2 Recursive Member Functions
  - The `locateNode()` Function
  - The `insert()` Function
  - The `remove()` Function
- 3 Assignment

# Assignment

## Assignment

- Read Section 20.5.