Recursive Linked Lists

Lecture 29
Section 10.3

Robb T. Koether

Hampden-Sydney College

Wed, Apr 1, 2009
1. Recursive Linked Lists
2. Recursive Member Functions
3. The Recursive Insert Function
4. Evaluation of List Implementations
5. Assignment
A linked list is a naturally recursive structure.
The “linked list” is a pointer to a node. (Ignore the mSize 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 mSize data member, so they are not exactly LinkedList objects.
Two-Part Implementation of Recursive Member Functions

Recursive Member Functions

```cpp
Type Class::func(parameters)
{
    // Do some special things...
    // Call the second function,
    // passing the head of the list
    func(head, parameters);
    // 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)

```c++
Type Class::Function(LinkedListNode* node, parameters) {
    if (condition) {
        // Handle the recursive case
        return Function(node->next, parameters);
    } else {
        // Handle the nonrecursive case
        return;
    }
}
```
Example

Example (The `insert()` Function)

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

```cpp
void insert(int pos, const T& value) {
    // Do something special
    assert(pos >= 0 && pos <= mSize);
    // Call the recursive function
    insert(head, pos, value);
    // Do something special
    mSize++;
    return;
}
```
The Recursive \texttt{insert()} Function

Example (The Recursive \texttt{insert()} Function)

- The recursive \texttt{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 `insert()` Function)

- The recursive case
  - Is distinguished by the condition that `pos > 0` (action at a later node).
  - Passes a pointer to the next node.
  - Decrement the value of `pos`. 
The Non-Recursive Case

Example (The Recursive `insert()` Function)

```c
if (pos > 0)
    insert(node->next, pos - 1, value);
```
The Non-Recursive Case

Example (The Recursive insert() Function)

- 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 \( \text{node} \) is modified, it must be passed by reference.
The Non-Recursive Case

Example (The Recursive insert() Function)

```cpp
if (pos == 0)
{
    LinkedListNode<T>* new_node
        = new LinkedListNode<T>(value);
    new_node->next = node;
    node = new_node;
}
```
The Recursive `insert()` Function

Example (Recursive `insert()` Function)

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

- Download `recurlinkedlist.h`
- Download and run `ListTest.cpp`
# Evaluation of Lists

<table>
<thead>
<tr>
<th>Type of List</th>
<th>get Element (seq)</th>
<th>get Element (rand)</th>
<th>insert</th>
<th>remove</th>
<th>push Front</th>
<th>pop Front</th>
<th>push Back</th>
<th>pop Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array List</td>
<td>0.002</td>
<td>0.025</td>
<td>6.55</td>
<td>8.53</td>
<td>12.4</td>
<td>16.2</td>
<td>0.020</td>
<td>0.013</td>
</tr>
<tr>
<td>Circ Array List</td>
<td>0.009</td>
<td>0.034</td>
<td>3.18</td>
<td>2.94</td>
<td>0.020</td>
<td>0.016</td>
<td>0.018</td>
<td>0.016</td>
</tr>
<tr>
<td>Linked List</td>
<td>8.19</td>
<td>7.72</td>
<td>33.9</td>
<td>36.9</td>
<td>0.095</td>
<td>0.077</td>
<td>16.8</td>
<td>33.5</td>
</tr>
<tr>
<td>Linked List w/Tail</td>
<td>8.46</td>
<td>7.97</td>
<td>34.3</td>
<td>37.2</td>
<td>0.098</td>
<td>0.083</td>
<td>0.094</td>
<td>16.8</td>
</tr>
<tr>
<td>Doubly Linked List</td>
<td>4.17</td>
<td>4.19</td>
<td>18.5</td>
<td>20.0</td>
<td>0.097</td>
<td>0.072</td>
<td>0.099</td>
<td>0.073</td>
</tr>
<tr>
<td>Circ Linked List</td>
<td>4.17</td>
<td>4.18</td>
<td>18.4</td>
<td>20.0</td>
<td>0.100</td>
<td>0.075</td>
<td>0.102</td>
<td>0.074</td>
</tr>
<tr>
<td>Recur Linked List</td>
<td>34.3</td>
<td>32.5</td>
<td>81.0</td>
<td>79.7</td>
<td>0.096</td>
<td>0.086</td>
<td>124</td>
<td>111</td>
</tr>
</tbody>
</table>
Assignment

**Homework**

- Read Section 10.3, pages 548 - 551.