List Iterator Implementation

Lecture 29
Section 14.6

Robb T. Koether

Hampden-Sydney College

Fri, Apr 6, 2018

Outline

Implementation

- Open `linkedlistwiter.h` and observe:
  - The constructor
  - `operator++()`
  - `operator*()`
  - `begin()`
  - `end()`

Outline

List Traversals

Definition (Traverse)

To traverse a list is to move systematically through its nodes, “visiting” each node along the way. Forward traversals go from head to tail. Reverse traversals go from tail to head.

- The meaning of “visiting” a node is left unspecified at this point.
- The meaning will be specified at the time of the traversal.

The Traversal Function

The traverse() Function

```
void traverse(void visit(Iterator&));
```

- Introduce a new List member function `traverse()`.
- The parameter `visit` is a pointer to a function.
- The `visit()` function has prototype

```
void visit(Iterator& it);
```

Traversals and Iterators

Traversal Implementation

```
void traverse(void visit(Iterator&))
{
    for (Iterator it = begin(); it != end(); ++it)
        visit(it);
    return;
}
```

- The `traverse()` function is implemented as a **`for`** loop.

Example - Print the List

The `print()` Function

```
list.traverse(print);
        ⋮
void print(Iterator& it)
{
    cout << *it << endl;
    return;
}
```

- For example, we could write a `print()` function and then use `traverse()` to print all the values in the list.

Outline

Reverse Iterators

- A reverse iterator is an iterator that advances in the opposite direction, from tail to head.
- It is initialized to the last element in the list.
- It "advances" until it has gone *beyond* the head of the list.
- Because a reverse iterator is an iterator, we will derive the `ReverseIterator` class from the `Iterator` class.

`ReverseIterator` Member Functions

Additional `ReverseIterator` Member Functions

```
ReverseIterator(const LinkedListwIter<T>* lst,
    LinkedListNode<T>* p);
ReverseIterator& operator++();
```

- `ReverseIterator(LinkedListwIter<T>*, LinkedListNode<T>*)` – Construct a `ReverseIterator`.
- **`operator`**`++()` – Advance the `ReverseIterator` to the next node.

`LinkedListwIter` Member Functions

Additional `LinkedListwIter` Member Functions

```
ReverseIterator rbegin() const;
ReverseIterator rend() const;
```

- `rbegin()` – Create a `ReverseIterator` set to the beginning (tail) of the list.
- `rend()` – Create a `ReverseIterator` set to the end (beyond the head) of the list.

`LinkedListwIter` Member Functions

- The other `LinkedListwIter` member functions that use iterators, such as the iterator version of `operator[]()`, can accept reverse iterators as well because

  A `ReverseIterator` IS-A `Iterator`.

Implemention of Reverse Iterators

- To construct a reverse iterator for a linked list,
  - Introduce a stack data member.
  - Push `NULL` onto the stack.
  - Then push the addresses of the nodes onto the stack as the list is traversed from head to tail.
  - Stop with all but the final `NULL` pointer on the stack.
  - Now the reverse iterator is initialized.

Implemention of Reverse Iterators

- To increment a reverse list iterator
  - Pop an address off the stack.
  - Assign it to the node pointer.
- To decrement a reverse list iterator
  - Push the current address onto the stack.
  - Use the `m_next` pointer as the current address.

Outline

Assignment

Assignment

- Read Section 14.6.