```
  friends.print();
  it.reset();                       // sets current to first item
  ++it;                             // sets current to second item
  it = "Davis, Jim";                // replace with new name
  ++it;                             // sets current to third item
  it.remove();                      // removes third item
  friends.print();
  if (!it) it.preInsert("Morse, Sam");
  friends.print();
  for (it.reset(); !it; ++it)       // traverses entire list
    it = "[" + it() + "]";
  friends.print();
}
Bowen, Van -> Dixon, Tom -> Mason, Joe -> White, Ann -> *
Bowen, Van -> Davis, Jim -> White, Ann -> *
Bowen, Van -> Davis, Jim -> Morse, Sam -> White, Ann -> *
[Bowen, Van] -> [Davis, Jim] -> [Morse, Sam] -> [White, Ann] -> *
```

The `for` loop changes each `data` value in the list by prepending a left bracket and appending a right bracket to each string. Note that the assignment `it = "[" + it() + "]"` calls the `operator()()` and `operator=()` functions of the `ListIter<string>` class as well as the constructor `string(const char*)` and `operator+=()` function defined in the `string` class.

To give `ListIter` objects the access to the `protected` members of `List` objects that they need to do their job, we need to declare the `ListIter` class a `friend` of the `List` class:

```
template<class T>
class List
{   friend class ListIter<T>;
  public:
    // other members
  protected:
    ListNode<T>* first;
    // other members
};
```

`List` iterators also need the access to the `protected` members of `ListNode` objects:

```
template<class T>
class ListNode
{   friend class List<T>;
    friend class ListIter<T>;
  public:
    ListNode(T& t, ListNode<T>* p) : data(t), next(p) { }
  protected:
    T data;               // data field
    ListNode* next;       // points to next node in list
};
```

An iterator acts like a window, allowing access to one item at a time in the container. Iterators are sometimes called *cursors* because they locate a specific element among the entire structure, the same way that a cursor on your computer screen locates one character location.

A structure may have more than one iterator. For example, one could declare three iterators on a list like this:

```
List<float> list;
ListIter<float> it1(list), it2(list), it3(list);
```