```
// FORWARD ITERATORS to step through the nodes of a linked list
// A node_iterator of can change the underlying linked list through the
// * operator, so it may not be used with a const node. The
// node_const_iterator cannot change the underlying linked list
// through the * operator, so it may be used with a const node.
// WARNING:
// This classes use std::iterator as its base class;
// Older compilers that do not support the std::iterator class can
// delete everything after the word iterator in the second line:

template <class Item>
class node_iterator
: public std::iterator<std::forward_iterator_tag, Item>
{
public:
    node_iterator(node<Item>* initial = NULL)
        { current = initial; }
    Item& operator *( ) const
        { return current->data( ); }
    node_iterator& operator ++( ) // Prefix ++
        {
            current = current->link( );
            return *this;
        }
    node_iterator operator ++(int) // Postfix ++
        {
            node_iterator original(current);
            current = current->link( );
            return original;
        }
    bool operator ==(const node_iterator other) const
        { return current == other.current; }
    bool operator !=(const node_iterator other) const
        { return current != other.current; }
private:
    node<Item>* current;
};

template <class Item>
class const_node_iterator
: public std::iterator<std::forward_iterator_tag, const Item>
{
public:
    const_node_iterator(const node<Item>* initial = NULL)
        { current = initial; }
    const Item& operator *( ) const
        { return current->data( ); }
    const_node_iterator& operator ++( ) // Prefix ++
        {
            current = current->link( );
            return *this;
        }
    const_node_iterator operator ++(int) // Postfix ++
        {
            const_node_iterator original(current);
            current = current->link( );
            return original;
        }
    bool operator ==(const const_node_iterator other) const
        { return current == other.current; }
    bool operator !=(const const_node_iterator other) const
        { return current != other.current; }
private:
    const node<Item>* current;
};
```