```
#include <cassert>
#include <climits>

template <typename NodeData>
class LinkedList
{
private:

    struct Node
    {
        NodeData data;  // "content" of node
        Node    *next;  // link to next node

        // Constructor functions
        Node(){}
        Node(const NodeData &theData, Node *const theNext = 0)
                : data(theData), next(theNext) {}
    };
public:
    // Constructor
    LinkedList() : first(0), current(0), pred(0) {}

    // True if list empty
    bool Empty() const {return first == 0;}
    // True if current position is beyond the last entry
    bool AtEnd() const {return current == 0;}
    // Rewind current entry to beginning of list
    void Rewind() {current = first; pred = 0;}
    // Skip to the next entry in the list
    void Skip();

    // Get the contents of the current list entry
    NodeData CurrentEntry() const
    {
        assert(!AtEnd());
        return current->data;
    }

    // Insert a new list entry before the current entry
    void Insert(const NodeData &d);
    // Update the current entry
    void Update(const NodeData &d) {assert(!AtEnd()); current->data = d;}
    // Delete the current entry
    // the new current entry is the successor of the deleted node
    void Delete();
private:
    Node *first;        // point to first node in list
    Node *current;      // point to the current node
    Node *pred;         // point to node preceding curent node
};

// move the current node in the list forward one node
template <typename NodeData>
void LinkedList<NodeData>::Skip()
{
    assert(!AtEnd());
    pred = current;
    current = current->next;
    pred->next = current;
}
```

```
// Insert a node in front of the current node
template <typename NodeData>
void LinkedList<NodeData>::Insert(const NodeData &d)
{
    Node *temp = new(nothrow) Node(d);
    assert(temp != NULL);

    // if first == current, list is either empty or you are entering node in front
    // of current. either way, first needs to be updated.
    if (first == current)
    {
        temp->next = first;
        first = temp;
    }
    // put new node in front of current node
    else
    {
        temp->next = current;
        pred->next = temp;
    }
    pred = temp;
}
template <typename NodeData>
void LinkedList<NodeData>::Delete()
{
    assert(!AtEnd());

    // if first == current, need to delete node and update first node
    if (first == current)
    {
        first = current->next;
        delete current;
        current = first;
    }
    // delete current node and make current successor node
    else
    {
        pred->next = current->next;
        delete current;
        current = pred->next;
    }
}

```