```
// Demonstrate use of Templates to make a linked list.
// ECE3090 - Spring 2012
// George F. Riley, Georgia Tech, Spring 2012

#include <iostream>

using namespace std;

template <typename T> class List;

// Define a templated class that contains the user’s data
// and a "next" pointer.
template <typename T> class ListNode
{
public:
  ListNode(const T& e) : next(0), element(e) {}
public:
  ListNode<T>* next;
  T            element;  // The user’s data
};

template <typename T> class ListIterator
{ // Define an "iterator" to access list elements
public:
  ListIterator() : current(0) {}
  ListIterator(ListNode<T>* b) :  current(b) {}

  // Define not equal operator
  bool operator !=(const ListIterator & rhs)
    {
      return current != rhs.current;
    }

  ListIterator operator++(int) // Postfix increment
    { // Postfix increment
      // Create a temporary to return the value prior to advance
      ListIterator tmp(*this);
      if  (current) current = current->next;
      return tmp;
    }

  ListIterator operator++()    // Prefix increment
    { // Prefix  increment
      if  (current) current = current->next;
      return *this;
    }

  T& operator*() const // Dereference operator
    {
      return current->element;
    }
private:
  ListNode<T>* current;
  friend class List<T>;
};
```

Program templatelinkedlist.cc

```
// Now define the "List" class
template <typename T> class List {
public:
  List() : head(0), tail(0) {}
  void PushBack(const T& e)
  { // Add to end
    ListNode<T>* n = new ListNode<T>(e); // Make a new node
    if (!head) head = n;                 // If list is empty, n is new head
    else tail->next = n;                 // Otherwise, old tail -> next is n
    tail = n;                            // n is always new tail
  }

  void PushFront(const T& e)
  { // Add to beginning
    ListNode<T>* n = new ListNode<T>(e); // Make a new node
    n->next = head;                      // New element next is old head
    if (!tail) tail = n;                 // List was empty, n is new tail
    head = n;                            // n is always new head
  }

  void Insert(const ListIterator<T>& i, const T& e)
    { // Insert an element after specified iterator
      ListNode<T>* n = new ListNode<T>(e);
      n->next = i.current->next;
      i.current->next = n;
      if (i.current == tail) tail = n;
    }


// For illustration, this is an implementation of "erase"
// for a doubly linked list.  It is not possible to efficiently
// erase an element in a singly linked list.  THis is commented out
// since we do not have a "prior" pointer in the singly-linked list.
//   void Erase(ListIterator<T> it)
//   {
//     ListNode<T>* p = it.current->prior;
//     ListNode<T>* n = it.current->next;
//     if (p)
//       { // Prior exists
//         p->next = n; // Prior->next points to current->next
//       }
//     else
//       { // The erased object is the old head, advance head
//         head = n;
//       }
//     if (n)
//       { // Next exists
//         n->prior = p;
//       }
//     else
//       { // The erased object is old tail, back up tail
//         tail = p;
//       }
//     (*it.current).~ListNode<T>(); // Call the destructor for erased object
//   }

```

Program templatelinkedlist.cc (continued)

```
  ListIterator<T> Begin()
  { // Return an iterator starting at the first element
    return ListIterator<T>(head);
  }

  ListIterator<T> End()
  { // Return an iterator representing one beyond end
    return ListIterator<T>(0);
  }

private:
  ListNode<T>* head;
  ListNode<T>* tail;
};

int main()
{
  List<int> l;
  for (int i = 0; i < 10; ++i) l.PushBack(i);
  for (int i = 100; i < 110; ++i) l.PushFront(i);
  ListIterator<int> it = l.Begin();
  ListIterator<int> i1;  // For inserting after specified element

  for (ListIterator<int> i = l.Begin(); i != l.End(); ++i)
    {
      cout << "List item is " << *i << endl;
      if ((*i) == 5) i1 = i; // Save this to insert after later
    }
  l.Insert(i1, 15);
  cout << "After insert" << endl;
  // Print the list again
  for (ListIterator<int> i = l.Begin(); i != l.End(); ++i)
    {
      cout << "List item is " << *i << endl;
    }
}
```

Program templatelinkedlist.cc (continued)