Swinburne University Of Technology

Faculty of Science, Engineering and Technology

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	6, Copy Control & Memory Management
Lecturer:	Dr. Markus Lumpe

Template class List

Add proper copy control to the template class `List`.

```
#pragma once

#include "DoublyLinkedNode.h"
#include "DoublyLinkedNodeIterator.h"
#include <stdexcept>

template<class T>
class List
{
private:
  // auxiliary definition to simplify node usage
  typedef DoublyLinkedNode<T> Node;

  Node* fTop;           // the first element in the list
  Node* fLast;          // the last element in the list
  int fCount;           // number of elements in the list

public:
  // auxiliary definition to simplify iterator usage
  typedef DoublyLinkedNodeIterator<T> Iterator;

  List();                     // default constructor - creates empty list
  ~List();                    // destructor - frees all nodes

  List( const List& aOtherList );              // copy constructor
  List& operator=( const List& aOtherList );   // assignment operator

  bool isEmpty() const;                        // Is list empty?
  int size() const;                            // list size

  // adds a node initialized with aElement at front
  void push_front( const T& aElement );

  // adds a node initialized with aElement at back
  void push_back( const T& aElement );

  // removes node that matches aElement from list
  void remove( const T& aElement );

  const T& operator[]( unsigned int aIndex ) const;   // list indexer

  // returns an iterator for the nodes of the list
  Iterator getIterator() const;
};
```

Test harness

Continue with the test code used in problem set 6. Add the following code:

```
List<string> copy( lList );

// iterate from the top
cout << "A - Top to bottom: " << copy.size() << " elements" << endl;

for ( List<string>::Iterator iter = copy.getIterator();
                        iter != iter.rightEnd(); iter++ )
{
      cout << "A list element: " << *iter << endl;
}

// override list
lList = copy;

lList.push_front( s6 );
lList.push_back( s1 );

// iterate from the top
cout << "B - Top to bottom: " << lList.size() << " elements" << endl;

for ( List<string>::Iterator iter = lList.getIterator().last();
                        iter != iter.leftEnd(); iter-- )
{
      cout << "A list element: " << *iter << endl;
}
```

The result should look like this:

```
A - Top to bottom: 3 elements
A list element: BBBB
A list element: DDDD
A list element: EEEE
B - Bottom to top: 5 elements
A list element: AAAA
A list element: EEEE
A list element: DDDD
A list element: BBBB
A list element: FFFF
```