Swinburne University Of Technology

Faculty of Information and Communication Technologies

ASSIGNMENT COVER SHEET

Subject Code:	HIT3303/8303
Subject Title:	Data Structures & Patterns
Assignment number and title:	5 – Copy Control and Queues
Due date:	**May 12, 2009, 02:30 p.m., on paper**
Lecturer:	Dr. Markus Lumpe

Your name:____________________

Marker's comments:

Problem	Marks	Obtained
1	24	
2	14	
Total	38	

Extension certification:

This assignment has been given an extension and is now due on _________________

Signature of Convener:________________________

Problem Set 3: Copy Control and Queues

Problem 1:

We start with the `List` data type for which we defined a bug fix in problem set 4. The specification of `List` does not yet address copy control and also lacks a facility to obtain the number of stored list nodes.

Implement the required modifications. Start with the solution of problem set 4 and verify your results with the supplied main test code.

The new specification for template class `List` is as follows:

```
template<class ElementType>
class List
{
private:

#include "NodeIterator.h"

  Node<ElementType>* fTop;
  Node<ElementType>* fLast;

  int fCount;

public:
  typedef NodeIterator<ElementType> ListIterator;

  List();
  List( const List<ElementType>& aOtherList );
  List<ElementType>& operator=( const List<ElementType>& aOtherList );
  ~List();

  bool IsEmpty() const;
  int Count() const;

  void Add( const ElementType& aElement );
  void AddFirst( const ElementType& aElement );
  bool Delete( const ElementType& aElement );
  void DeleteFirst();
  void DeleteLast();

  const ElementType& operator[]( int aIndex ) const;
  ListIterator GetIterator() const;
};
```

Test:

```
void test1()
{
  string s1( "One" );
  string s2( "Two" );
  string s3( "Three" );
  string s4( "Four" );

  List<string> l;

  l.Add( s1 );
  l.Add( s2 );
  l.Add( s3 );
  l.Add( s4 );

  cout << "The list:" << endl;
  for ( List<string>::ListIterator iter = l.GetIterator();
                                          iter != iter.end(); iter++ )
    cout << *iter << endl;

  List<string> c1 = l;

  cout << "The first copy:" << endl;
  for ( List<string>::ListIterator iter = c1.GetIterator();
                                          iter != iter.end(); iter++ )
    cout << *iter << endl;

  List<string> c2;
  c2 = l;

  cout << "The second copy:" << endl;
  for ( List<string>::ListIterator iter = c2.GetIterator().end();
                                          --iter != iter.begin(); )
    cout << *iter << endl;
}
```

Output:

```
The list:
One
Two
Three
Four
The first copy:
One
Two
Three
Four
The second copy:
Four
Three
Two
One
```

Problem 2:

A stack manages elements in the last-in, first-out manner (LIFO). Stacks have frequent application. We find stacks almost everywhere in computing.

Consider the following specification for template class Stack:

```
template<class T>
class Stack
{
private:
  List<T> fContents;

public:
  Stack();
  ~Stack();

  bool IsEmpty() const;
  int Size() const;

  void Push( const T& aElement );
  void Pop();
  const T& Top() const;

  const T& operator[]( int aIndex ) const;
};
```

The template class Stack specifies the behavior of a stack. Moreover, through the use of List, we guarantee proper copy control for Stacks and we can therefore rely on the default, C++-generated mechanisms.

Implement the required member functions. Use the solution of problem 1 and verify your results with the supplied main test code.

Test:

```
void test2()
{
  string sa( "One" );
  string sb( "Two" );
  string sc( "Three" );
  string sd( "Four" );

  Stack<string> s1;

  s1.Push( sa );
  s1.Push( sb );
  s1.Push( sc );
  s1.Push( sd );

  cout << "The stack s1:" << endl;
  for ( int i = 0; i < s1.Size(); i++ )
    cout << s1[i] << endl;

  Stack<string> c1 = s1;

  cout << "The first copy:" << endl;
  for ( int i = 0; i < c1.Size(); i++ )
    cout << c1[i] << endl;

  Stack<string> c2;
  c2 = s1;

  cout << "The second copy:" << endl;
  for ( int i = 0; i < c2.Size(); i++ )
    cout << c2[i] << endl;

  Stack<string> s2;

  while ( !s1.IsEmpty() )
  {
    s2.Push( s1.Top() );
    s1.Pop();
  }

  cout << "The stack s2:" << endl;
  for ( int i = 0; i < s2.Size(); i++ )
    cout << s2[i] << endl;
}
```

Output:

```
The stack s1:
Four
Three
Two
One
The first copy:
Four
Three
Two
One
The second copy:
Four
```

```
Three
Two
One
The stack s2:
One
Two
Three
Four
```

Submission deadline: Tuesday, May 12, 2009, 2:30 p.m.

Submission procedure: on paper in class.