CMPUT 115 Section B1
Term Test 2

March 12, 2001

Instructions:

- This is a closed book, no notes exam.
- Try to put all of your answers in the space provided.
- There are some blank pages at the end of the booklet for use as scrap paper.
- Please do not open the exam until you are instructed to do so.
- Good luck.

First Name:

Last Name:

1. [10 Marks] Here is some of the List interface.

```
public interface List extends Collection
{
    public Iterator elements();
    // post: returns an iterator allowing
    //    ordered traversal of elements in list

    public boolean isEmpty();
    // post: returns true iff list has no elements

    public void addToHead(Object value);
    // post: value is added to beginning of list

    public void addToTail(Object value);
    // post: value is added to end of list

    public Object removeFromHead();
    // pre: list is not empty
    // post: removes first value from the list

    public Object removeFromTail();
    // pre: list is not empty
    // post: removes the last value from the list

    // etc ...
}
```

The three implementations of the List interface that we studied in class (SinglyLInkedList, DoublyLinkedList and CircularList) are all linked structures. Could the List interface be implemented without using a linked structure? If so how? If not then why not?

2. [15 Marks] What is the time complexity of the following code fragment? Express your answer as a function of **N**, and be as exact as you can be. Some of the source code for the OrderedVector class can be found at the end of this exam booklet.

```
OrderedVector v = new OrderedVector();
for ( int i = 0; i < N; i++ )
  v.add( new Integer(i) );
```

3. [15 Marks] What output would the following code produce? Please put your answer to the right of the code.

```
public static void output()
{
  Stack stack = new StackList();
  Queue queue = new QueueList();

  for ( int i = 0; i < 5; i++ )
    stack.add( new Integer(i) );

  while ( !stack.isEmpty() )
    queue.add( stack.remove() );

  while ( !queue.isEmpty() )
    System.out.println( queue.remove() );
}
```

4. [15 Marks] How many calls to the String class's *compareTo* method will be made when the following code fragment is executed?

```
OrderedList list = new OrderedList();
list.add( "A" );
list.add( "C" );
list.add( "E" );
list.add( "B" );
list.add( "D" );
list.add( "F" );
```

5. [20 Marks] The interface java.util.Iterator is defined as follows (the main difference from structure.Iterator is the remove method).

```
public interface Iterator {
  public boolean hasNext();
  public Object next();
  public void remove();
}
```

Complete the following implementation of the VectorIterator class which implements the java.util.Iterator interface (instead of structure.Iterator). To do this you will need to implement the **remove** method. There is space to put your answer on the next page.

```
public class VectorIterator implements java.util.Iterator {
  protected Vector theVector;
  protected int current;

  public VectorIterator(Vector v) {
  // Constructs an initialized iterator associated with v.
    theVector = v;
    current = 0;
  }

  public boolean hasNext() {
  // Returns true if the iteration has more elements.
    return current < theVector.size();
  }

  public Object next() {
  // Returns the next element in the iteration
    return theVector.elementAt(current++);
  }

  public void remove() {
  // Removes from the underlying collection the last element
  // returned by the iterator. Calling this method should not
  // affect what is returned by the next call to next().
```

6. [15 Marks] Complete the following implementation of the OrderedList's *add* method by filling in the blanks.

```
public class OrderedList implements OrderedStructure
{
    protected SinglyLinkedListElement data; // smallest value
    protected int count;        // number of values in list

    public void add(Object value)
    // pre: value is non-null
    // post: value is added to the list, leaving it in order
    {
        SinglyLinkedListElement previous = null;
        SinglyLinkedListElement finger = data;
        Comparable cValue = (Comparable)value;
        // search for the correct location



        // spot is found, insert



        count++;
    }
```

Some source code from the structure package.

```
public class OrderedVector implements OrderedStructure
{
    protected Vector data;

    public void add(Object value)
    // pre: value is non-null
    // post: inserts value, leaves vector in order
    {
        int position = indexOf((Comparable)value);
        data.insertElementAt(value,position);
    }

    protected int indexOf(Comparable target)
    // pre: target is a non-null comparable object
    // post: returns ideal position of value in vector
    {
        Comparable midValue;
        int low = 0;  // lowest possible location
        int high = data.size(); // highest possible location
        int mid = (low + high)/2; // low <= mid <= high
        while (low < high) {
            midValue = (Comparable)data.elementAt(mid);
            if (midValue.compareTo(target) < 0) {
                low = mid+1;
            } else {
                high = mid;
            }
            mid = (low+high)/2;
        }
        return low;
    }

    // etc ...

} // end of OrderedVector class
```

Student Id Number: ____________________

Extra paper for rough work

Student Id Number: ____________________

Extra paper for rough work