CMPUT 115 Section B3
Term Test 2

March 13, 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:

Student Id Number: _____

1. [10 Marks] In what way is the *List based* implementation of the Queue interface better than the *Vector based* implementation?

2. [10 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 SinglyLinkedList class can be found at the end of this exam booklet.

```
SinglyLinkedList list = new SinglyLinkedList();
for ( int i = 0; i < N; i++ )
  list.addToTail( new Integer(i) );
```

3. [15 Marks] What output would the following code produce? Some of the source code for the SinglyLinkedListIterator class can be found at the end of this exam booklet.

```
SinglyLinkedList list = new SinglyLinkedList();
for ( int i = 0; i < 10; i++ )
  list.addToHead( new Integer(i) );

Iterator it = list.elements();
while ( it.hasMoreElements() )
  System.out.println( it.nextElement() );

list.removeFromHead();
list.removeFromHead();
list.removeFromHead();
list.remove( new Integer(1) );

System.out.println( "-----" );

it.reset();
while ( it.hasMoreElements() )
  System.out.println( it.nextElement() );
```

4. [15 Marks] How many calls to the String class's *equals* method will be made when the following code fragment is executed? Put your answer to the right of the code.

```
Vector v = new Vector();
v.addElement( "a" );
v.addElement( "c" );
v.addElement( "c" );
v.addElement( "e" );
v.addElement( "b" );
v.addElement( "d" );
v.addElement( "f" );
v.removeElement( "f" );
v.removeElement( "f" );
v.removeElement( "c" );
```

5. [15 Marks] Complete the following implementation of OrderedVector's *indexOf* method.

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

    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 =
        int high =
        int mid =

        while (low < high) {









    }
```

6. [15 Marks] Complete the following implementation of a *bi-directional* iterator for the DoublyLinkedList class. A bi-directional iterator is one that has a *previousElement* method. The previousElement method should do the same thing as the nextElement method except that instead of incrementing the iterator it should decrement it. That is, it does the same thing but in the opposite direction.

```
public class DoublyLinkedListBiIterator implements Iterator {
    protected DoublyLinkedListElement head;
    protected DoublyLinkedListElement current;

    public DoublyLinkedListIterator(DoublyLinkedListElement h) {
    // post: constructs an iterator rooted at list head, h
        head = h;
        reset();
    }

    public void reset() {
    // post: resets iterator to list head
       current = head;
    }

    public boolean hasMoreElements() {
    // post: returns true iff current element is valid
        return current != null;
    }

    public Object nextElement() {
    // post: returns current element and increments iterator
        Object result = current.value();
        current = current.next();
        return result;
    }

    public Object previousElement() {
    // post: returns current element and decrements iterator




    }
}
```

Some source code from the structure package.

```
public class SinglyLinkedList implements List {
    protected int count;                        // list size
    protected SinglyLinkedListElement head; // first elt

    public void addToTail(Object value) {
    // post: adds value to end of list
        // location for the new value
        SinglyLinkedListElement temp =
            new SinglyLinkedListElement(value,null);
        if (head != null) {
            // pointer to possible tail
            SinglyLinkedListElement finger = head;
            while (finger.next() != null) {
                finger = finger.next();
            }
            finger.setNext(temp);
        } else head = temp;
        count++;
    }
    // etc ...

} // end of SinglyLinkedList class

class SinglyLinkedListIterator implements Iterator {
    protected SinglyLinkedListElement current;
    protected SinglyLinkedListElement head;

    public SinglyLinkedListIterator(SinglyLinkedListElement t) {
    // post: returns an iterator that traverses a linked list.
        head = t;
        current = head;
    }

    public boolean hasMoreElements() {
    // post: returns true iff there are unvisited elements
        return current != null;
    }

    public Object nextElement() {
    // pre: hasMoreElements()
    // post: returns value and advances iterator
        Object temp = current.value();
        current = current.next();
        return temp;
    }
    // etc ...

} // end of SinglyLinkedListIterator class
```

Student Id Number: _____

Extra paper for rough work

Student Id Number: _____

Extra paper for rough work