COP 3337
Programming II

Examination 3

Name: ________________________________

November 30, 2006

This exam has 3 additional pages. Please answer each question on the page on which it is asked. You may write on the back of the facing page if you need to.
1. [40 pts] Consider the following partially implemented singly-linked list class and accompanying node class (declared in the same Java source file):

```java
class Node {
    Node( int d, Node n ) {
        data = d; next = n; }
    int data;
    Node next;
}

class LinkedList {
    private Node first;
    private Node last;

    public LinkedList( ) {
        first = last = null; }

    public void addFirst( int x) {
        /* TODO: Part (a) */

        // Throws IllegalArgumentException if already empty
    public void removeLast( ) {
        /* TODO: Part (b) */

    public String toString( ) {
        /* TODO: Part (c) */
    }
}
```

As the constructor shows, there are no header or tail nodes. If the list is empty, both `first` and `last` must be null.

(a) Implement `addFirst`, that adds `x` to the front of the list. Note that if the list is empty, you have a special case.

(b) Implement `removeLast`, which removes the last item from the list. If the list is empty, throw an exception. Note that if the list has only one item, you have a special case.

(c) Implement a reasonable `toString` that returns a `String` representation of the list, enclosed in brackets.
2. [40 pts] A simplified version of the IntMultiSet interface is as follows:

```java
public interface IntMultiSet
{
    public void add( int x );
    public boolean contains( int x );
    public boolean removeOne( int x ); // Returns true if remove succeeds
}
```

This question requires you to complete a class that implements this interface below. As shown, the data will use a `java.util.Map` in which the key is an item in the multiset, and the corresponding value is the number of times that the item occurs.

```java
public class MapIntMultiSet implements IntMultiSet {
    public boolean contains( int x )
    { /* TODO: Part (a) */ }

    public void add( int x )
    { /* TODO: Part (b) */ }

    public boolean removeOne( int x )
    { /* TODO: Part (c) */ }

    private Map<Integer,Integer> theMap = new TreeMap<Integer,Integer>();
}
```

(a) Implement `contains`.
(b) Implement `add`.
(c) Implement `removeOne`. 
3. [20 pts] A non-contiguous substring of String s is a sequence of \( k \geq 0 \) characters in s, in the order in which they occur in s. For instance, the following table shows ALL the non-contiguous substrings of some String s.

<table>
<thead>
<tr>
<th>String s</th>
<th>All non-contiguous substrings of s</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>&quot;ab&quot;</td>
<td>&quot;a&quot; &quot;b&quot; &quot;ab&quot;</td>
</tr>
<tr>
<td>&quot;abc&quot;</td>
<td>&quot;a&quot; &quot;b&quot; &quot;ab&quot; &quot;c&quot; &quot;ac&quot; &quot;bc&quot; &quot;abc&quot;</td>
</tr>
</tbody>
</table>

Observe that a String of length \( n \) has \( 2^n \) non-contiguous substrings. Write a routine generateAll that returns a List containing all the non-contiguous substrings of parameter s. Your solution will require the use of recursion. Implement generateAll below:

```java
public static List<String> generateAll(String s) {
}
```