1. Data Structures (20 Points)
   a. Explain why you may need to add a stack data structure to your code when you re-implement a recursive method without using recursion.

   b. Do you need to add a stack when you re-implement a method with “tail recursion”? (Y/N) _____
   Explain why or why not.
2. Collection Class/Interface Hierarchy (20 Points)

For each of the following, indicate if it is (a) valid Java statement(s) or not and explain.
(Assume that we have included any needed import statements for the java.util classes.)

<table>
<thead>
<tr>
<th>Statement(s)</th>
<th>Valid(Y/N)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Iterator&lt;ArrayList&gt; a = new ArrayList&lt;ArrayList&gt;();</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>b. Iterator&lt;String&gt; b = new Iterator&lt;String&gt;();</td>
<td>N</td>
<td>Cannot cast to Iterator&lt;String&gt;</td>
</tr>
<tr>
<td>c. ArrayList&lt;int&gt; c = new ArrayList&lt;int&gt;();</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>d. Collection&lt;String&gt; d = new ArrayList&lt;String&gt;(); if (d.size() != 0) some valid Java statement;</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
| e. public class SomeCollection<T> implements Iterable<T> {
   . . .
} | Y | |

Elsewhere:
   SomeCollection<String> s = new SomeCollection<String>();
   for (String word : s) some valid Java statement using word; | Y | |
3. Data Structures and their Implementation (30 Points)
a. Show the contents of an array used to implement a binary search tree using the computational strategy when the elements A-G are added in the following order: C, A, E, D, G, B, and F. The letter indicates the natural ordering of the objects.

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
</tr>
</tbody>
</table>

b. To remove node E, what rule would you use to replace it and why?

c. Show the steps required to remove the object for the letter E from the array and maintain the search tree integrity. (Note: You may not need all of the following lines.)

<table>
<thead>
<tr>
<th>Element</th>
<th>from Index</th>
<th>to Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>______</td>
<td>None (Removed from Tree)</td>
</tr>
<tr>
<td></td>
<td>______</td>
<td></td>
</tr>
<tr>
<td></td>
<td>______</td>
<td></td>
</tr>
<tr>
<td></td>
<td>______</td>
<td></td>
</tr>
<tr>
<td></td>
<td>______</td>
<td></td>
</tr>
<tr>
<td></td>
<td>______</td>
<td></td>
</tr>
<tr>
<td></td>
<td>______</td>
<td></td>
</tr>
</tbody>
</table>

d. If you add E back after removing it, it would be added as the ______ child of node ____.

e. Show the state of the array after removing E from the tree and re-adding it.

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
<td>___</td>
</tr>
</tbody>
</table>
4. Binary Tree Level Order Iterator (30 Points)
Given that you have a generic binary tree implemented in an array according to the class on
the next page, write the code for an IteratorLevelOrder class that performs a level order
traversal of the elements in the tree. The remove method does not need to be implemented.

In file IteratorLevelOrder.java:
You may tear this page off your exam to use in Problem 4.

In file BinaryTree.java:
import java.util.*;

public class BinaryTree<T> implements Iterable<T>
{
    private T [] tree; // using computational strategy
    private int count;

    public BinaryTree(int size)
    {
        tree = (T []) new Object[size];
        count = 0;
    }

    // code for add, remove, etc. not shown

    public Iterator<T> iterator()
    {
        return new IteratorLevelOrder<T>(tree, count);
    }
}
1. Data Structures
   a. When a recursive method executes, the state of its variables at each level in the sequence of recursive calls is kept on the system stack. They are pushed on the system stack each time the method calls itself and they are popped from the system stack each time the recursive method returns.

   When you rewrite the recursive method without recursion, if there are any variables that need to be preserved from the first part of one iteration of the loop, reused during the next iteration of the loop, and returned to their original state upon resumption of the rest of the previous iteration, the code in the method must explicitly push those variables on a local stack and pop them off at the correct places in the code. The code simulates the behavior of the system stack.

   b. No. You do not need a stack when you re-implement a tail recursive method without recursion because the recursive call is the last action in a tail recursive method and the variables at each level in the recursive calls are never used again after the recursive call returns. Even though the recursive calls preserve the state of these variables, there is no need for it. When you modify the code to use a loop, you do not need to restore the state of the variables from a previous iteration after any subsequent iteration.

2. Collection Class/Interface Hierarchy
   Statement(s)   Valid(Y/N)   Explanation
   a. Iterator<ArrayList> a = new ArrayList<ArrayList>();
      ___No___  ArrayList does not implement Iterator

   b. Iterator<String> b = new Iterator<String>();
      ___No___  Iterator is an interface and can not be instantiated

   c. ArrayList<char> c = new ArrayList<char>();
      ___No___  char is a primitive type not a reference type

   d. Collection<String> d = new ArrayList<String>();
      if (d.size() != 0)
         some valid Java statement;
      ___Yes___  ArrayList implements Collection and size()is OK

   e. public class SomeCollection<T> implements Iterable<T>
      {
         . . .
      }

   Elsewhere:
      SomeCollection<String> s = new SomeCollection<String>();
      for (String word : s)
         some valid Java statement using word;
      ___Yes___  "for each" works on classes that implement Iterable
3. Data Structures and their Implementation

b. Node E had two children so you replace it with its in order successor F.

c. Show the steps required to remove the object for the letter E and maintain the search tree integrity.

<table>
<thead>
<tr>
<th>Element</th>
<th>from Index</th>
<th>to Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>2</td>
<td>None (Removed from Tree)</td>
</tr>
<tr>
<td>F</td>
<td>13</td>
<td>2</td>
</tr>
</tbody>
</table>

No other nodes need to be moved.

d. If you add E back after removing it, it would be added as the right child of node D.

e. Show the state of the tree after removing E and re-adding E.

4. A level order traversal of a tree stored in an array is just in sequence through the array! However, null elements in the array are not included in the count. You have to base the return value of the hasNext method on the remaining count being greater than zero and to decrement the count in the next method only when returning a non-null value.

```java
import java.util.Iterator;
public class IteratorLevelOrder<T> implements Iterator<T> {
    private T[] tree;
    private int count;
    private int cursor;

    public IteratorLevelOrder(T[] tree, int count) {
        this.tree = tree;
        this.count = count;
        cursor = 0;
    }

    public boolean hasNext() {
        return count > 0;
    }

    public T next() {
        if (tree[cursor] != null) {
            count--;
            return tree[cursor++];
        }
    }
}
```