1. Binary Tree Traversals, 6 points. Consider the following tree.

```
+----+
 | 0  |
+----+
 /    |
+----+   +----+
 | 3 |     | 2 |
+----+     +----+
 /          /    
+----+      +----+      +----+
 | 7 |      | 9 |      | 5 |
+----+      +----+      +----+
 /          /    
+----+      +----+      +----+      +----+
 | 4 |      | 6 |      | 1 |      | 8 |
+----+      +----+      +----+      +----+
```

Fill in each of the traversals below:

**Preorder traversal**
__________________________________________________

**Inorder traversal**
__________________________________________________

**Postorder traversal**
__________________________________________________

2. Binary Search Tree, 4 points. Draw a picture below of the binary search tree that would result from inserting the following words into an empty binary search tree in the following order: Picard, Riker, Worf, Crusher, Troi, LaForge, Data. Assume the search tree uses alphabetical ordering to compare words.
3. Collections Programming, 15 points. Write a method called acronyms that takes a set of word lists as a parameter and that returns a map whose keys are acronyms and whose values are the word lists that produce that acronym. Acronyms are formed from each list as described in problem 4. Recall that the list [laughing, out, loud] produces the acronym "LOL". The list [League, of, Legends] also produces the acronym "LOL". Suppose that a variable called lists stores this set of word lists:

[[attention, deficit], [Star, Trek, Next, Generation],
 [laughing, out, loud], [International, Business, Machines],
 [League, of, Legends], [anno, domini], [art, director],
 [Computer, Science and, Engineering]]

Each element of this set is a list of values of type String. You may assume that each list is nonempty and that each string in a list is nonempty.

Your method should construct a map whose keys are acronyms and whose values are sets of the word lists that produce that acronym. For example, the call acronyms(lists) should produce the following map:

{AD=[[attention, deficit], [anno, domini], [art, director]],
 CSE=[[Computer, Science and, Engineering]],
 IBM=[[International, Business, Machines]],
 LOL=[[laughing, out, loud], [League, of, Legends]],
 STNG=[[Star, Trek, Next, Generation]]}

Notice that there are 5 unique acronyms produced by the 8 lists in the set. Each acronym maps to a set of the word lists for that acronym. Your method should not make copies of the word lists; the sets it constructs should store references to those lists. As in the example above, the keys of the map that you construct should be in sorted order. You may assume that you have available to you a method called acronymFor that takes a list of strings as a parameter and that returns the acronym for that list. Your method is not allowed to change either the set passed as a parameter or the lists within the set.
4. Comparable class, 15 points. Define a class called ClockTime that stores information about time of day using a standard clock. Each ClockTime object keeps track of hours, minutes, and a String to indicate "am" or "pm". It has the following public methods:

```java
ClockTime(hours, minutes, amPm)    constructs a ClockTime with given hours, minutes and amPm setting
getHours()                         returns the hours
getMinutes()                       returns the minutes
getAmPm()                          returns the am/pm setting
toString()                         returns a String representation of the time
```

Assume that the values passed to your constructor are legal. In particular, hours will be between 1 and 12 inclusive, minutes will be between 0 and 59 inclusive, and the am/pm parameter will be either the String "am" or the String "pm". These values should be returned by the various "get" methods.

The toString method should return a String composed of the hours followed by a colon followed by the minutes (2 digits) followed by a space followed by the am/pm String. For example, given these declarations:

```java
ClockTime time1 = new ClockTime(8, 31, "am");
ClockTime time2 = new ClockTime(12, 7, "pm");
```

time1.toString() should return "8:31 am" and time2.toString() should return "12:07 pm". You must exactly reproduce the format of these examples.

Your class should implement the Comparable<E> interface with earlier times considered "less." The earliest time is 12:00 am and the latest time is 11:59 pm. In between the time increases as it would in a standard clock. Keep in mind that 12:59 am is followed by 1:00 am, that 11:59 am is followed by 12:00 pm, and that 12:59 pm is followed by 1:00 pm.
5. Binary Trees, 10 points. Write a method called `oddPathSum` that returns the number of nodes in a tree that have an odd path sum. The path sum of a node is the sum of all values from the overall root to that node. For example if the variable `t` refers to the following tree:

```
+----+
| 5  |
+----+
      /\  \
     +----+
     | 3  |   | 12 |
     +----+   +----+
      /      /    / \
     +----+   +----+
     | 1  |   | 2  |   | 7  |
     +----+   +----+   +----+
      /      /    /      /    / \
     +----+   +----+   +----+   +----+
     | 6  |   | 11 |   | 10 |   | 4  |
     +----+   +----+   +----+   +----+
```

Then `t.oddPathSum()` should return 4 indicating that 4 of these nodes have an odd path sum (the nodes storing 5, 12, 1, and 6).

You are writing a public method for a binary tree class defined as follows:

```java
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    <constructors>
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

You are writing a method that will become part of the `IntTree` class. You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class. You may not construct any extra data structures to solve this problem.
6. Details of inheritance, 10 points. Assuming that the following classes have been defined:

```java
public class Couch extends Table {
    public void method1() {
        System.out.println("Couch 1");
    }
}

public class Table extends Chair {
    public void method1() {
        System.out.println("Table 1");
    }

    public void method2() {
        System.out.println("Table 2");
        super.method2();
    }

    public void method3() {
        System.out.println("Table 3");
        method1();
    }
}

public class Chair {
    public void method2() {
        System.out.println("Chair 2");
    }
}

public class Lamp extends Chair {
    public void method1() {
        System.out.println("Lamp 1");
    }

    public void method2() {
        System.out.println("Lamp 2");
    }
}
```
And assuming the following variables have been defined:

```java
Table var1 = new Table();
Chair var2 = new Table();
Table var3 = new Couch();
Chair var4 = new Lamp();
Chair var5 = new Couch();
Object var6 = new Chair();
```

In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, fill in the right-hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1.method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var2.method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var3.method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var4.method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var5.method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var6.method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var1.method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var2.method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var3.method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var4.method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var1.method3();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var2.method3();</td>
<td>__________________________</td>
</tr>
<tr>
<td>var3.method3();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Lamp)var4).method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Lamp)var2).method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Table)var5).method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Couch)var1).method1();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Chair)var6).method2();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Couch)var5).method3();</td>
<td>__________________________</td>
</tr>
<tr>
<td>((Table)var5).method3();</td>
<td>__________________________</td>
</tr>
</tbody>
</table>
7. Binary Trees, 20 points. Write a method called makeFull that turns a binary tree of integers into a full binary tree. A full binary tree is one in which every node has either 0 or 2 children. Your method should produce a full binary tree by replacing each node that has one child with a new node that has the old node as a leaf where there used to be an empty tree. The new node should store a value that indicates the level of the tree (-1 for the first level of the tree, -2 for the second level of the tree, and so on). For example, if a tree called t stores the following:

```
+-----+
 | 12 |
+-----+
 /     |
+-----+
 | 29 |
+-----+
```

and we make the call:
```
t.makeFull();
```
then the tree should store the following after the call:

```
+-----+
 | -1 |
+-----+
 /    /
+-----+  +-----+
 | 29 |   | 12 |
+-----+  +-----+
```

Notice that the node storing 12 that used to be at the top of the tree is now a leaf where there used to be an empty tree. In its place at the top of the tree is a new node that stores the value -1 to indicate that it was added at level 1 of the tree. Your method should perform this operation at every level of the tree. For example, if t had instead stored:

```
+-----+
 | 12 |
+-----+
```

then after the call it would store:

```
+-----+
 | 12 |
+-----+
```

```
+-----+
 | 28 |   | 19 |
+-----+  +-----+
 /     /
+-----+  +-----+
 | 94 |   | 32 |
+-----+  +-----+
```

```
+-----+  +-----+  +-----+  +-----+
 | 65 |   | 18 |   | 72 |
+-----+  +-----+  +-----+  +-----+
```

```
+-----+  +-----+  +-----+  +-----+  +-----+
 | 94 |   | 28 |   | -3 |   | 19 |
+-----+  +-----+  +-----+  +-----+  +-----+
```

```
+-----+  +-----+  +-----+  +-----+  +-----+
 | 65 |   | 18 |   | 32 |   | 72 |
+-----+  +-----+  +-----+  +-----+  +-----+
```
Notice that two nodes were added at level 2, and one at level 3.

You are writing a public method for a binary tree class defined as follows:

```java
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    // post: constructs an IntTreeNode with the given data and links
    public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}
```

```java
public class IntTree {
    private IntTreeNode overallRoot;

    // methods
}
```

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. YOU ARE NOT TO CHANGE THE DATA FIELD OF THE EXISTING NODES IN THE TREE (what we called "morphing" in assignment 10) and you are not to replace any existing nodes. You will, however, construct new nodes containing negative values to be inserted into the tree (notice that there is only one constructor for nodes). You will also change the links of the tree to restructure the tree as described. Your solution must run in O(n) time where n is the number of nodes in the tree.

8. Linked Lists, 20 points. Write a method of the LinkedIntList class called removeAlternatePairs that removes alternate pairs from a list of numbers, constructing and returning another LinkedIntList that contains the values removed. For example, suppose that a variable called list stores the following values:

```
[1, 2, 3, 4, 5, 6, 7, 8]
| | | | | | |
+---+---+---+---
pair pair pair pair
```

As indicated, this list has four pairs. If the following call is made:

```java
LinkedIntList result = list.removeAlternatePairs();
```

then after the call, list and result would store the following values:

```
list:   [3, 4, 7, 8]
result: [1, 2, 5, 6]
```

Notice that the first pair has been moved to the new list, the second pair has not been moved, the third pair has been moved, and the fourth pair has not been moved. This pattern should be repeated with every other pair from the original list being moved to the new list.
This example purposely used sequential integers to make the rearrangement clear, but you should not expect that the list will store sequential integers. It might end with a pair that is to be moved or, as in the example above, it might end with a pair that is not supposed to be moved. It also might end with an extra value that is not part of a pair. Only complete pairs should be moved to the new list. For example, if the list had stored this sequence of values:

```
[5, 17, 8, 2, 9, 42, 37, 23, 19, -6, -5, 0, 4]
```

<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>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>pair</td>
<td>pair</td>
<td>pair</td>
<td>pair</td>
<td>pair</td>
<td>pair</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

then after the call list and result would store the following values:

```
list: [8, 2, 37, 23, -5, 0, 4]
result: [5, 17, 9, 42, 19, -6]
```

Notice that the final pair of -5 and 0 is not moved but the final value of 4 is also not moved because it is not part of a complete pair. If no pairs are removed, the method should return an empty list.

You are writing a public method for a linked list class defined as follows:

```java
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

You are writing a method that will become part of the LinkedIntList class. You will need to call the zero-argument LinkedIntList constructor to construct the list to be returned and you may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. You are allowed to define your own variables of type ListNode, but you may not construct any new nodes, you may not use any auxiliary data structure to solve this problem (no array, ArrayList, stack, queue, String, etc), and your solution must run in O(n) time where n is the number of nodes in the list. You also may not change any data fields of the nodes. You MUST solve this problem by rearranging the links of the list.