1. Binary Tree Traversals. Consider the following tree.

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

Fill in each of the traversals below:

Preorder traversal ____

Inorder traversal   ____

Postorder traversal ____

2. Binary Search Tree. 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:

   Grumpy, Doc, Dopey, Bashful, Happy, Sneezy, Sleepy.

Assume the search tree uses alphabetical ordering to compare words.

3. Details of Inheritance. Assuming that the following classes have been defined:

```
        public class Cup extends Box {
            public void method1() {
                System.out.println("Cup 1");
            }

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

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

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

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

        public class Box extends Pill {
            public void method2() {
                System.out.println ("Box 2");
            }

            public void method3() {
                method2();
                System.out.println ("Box 3");
            }
        }
```

And assuming the following variables have been defined:

```
        Box var1 = new Box();
        Pill var2 = new Jar();
        Box var3 = new Cup();
        Box var4 = new Jar();
        Object var5 = new Box();
        Pill var6 = new Pill();
```

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.

Statement	Output
var1.method2();	______________________________
var2.method2();	______________________________
var3.method2();	______________________________
var4.method2();	______________________________
var5.method2();	______________________________
var6.method2();	______________________________
var1.method3();	______________________________
var2.method3();	______________________________
var3.method3();	______________________________
var4.method3();	______________________________
((Cup) var1).method1();	______________________________
((Jar) var2).method1();	______________________________
((Cup) var3).method1();	______________________________
((Cup) var4).method1();	______________________________
((Jar) var4).method2();	______________________________
((Box) var5).method2();	______________________________
((Pill) var5).method3();	______________________________
((Jar) var2).method3();	______________________________
((Cup) var3).method3();	______________________________
((Cup) var5).method3();	______________________________

4. **Collections.** Write a method called indexMap that takes a list of strings as a parameter and that returns a map that associates each string from the list to a list of indexes at which that string occurs. For example, if a variable called list stores the following:

```
[to, be, or, not, to, be]
```

then the call indexMap(list) should return the following map:

```
{be=[1, 5], not=[3], or=[2], to=[0, 4]}
```

Notice that each string from the original list maps to a list of indexes. For example, the string "to" appeared at index 0 and 4 in the original list, so it maps to a list with the values 0 and 4. The map returned by your method should be ordered alphabetically and the lists that each string maps to should have indexes in increasing order, as in the example.

Your method should construct the new map and each of the lists contained in the map and can construct iterators but should otherwise not construct any new data structures. It should also not modify the list of words passed as a parameter and it should be reasonably efficient.

Space for #4

5. **Comparable.** Define a class TeamData that keeps track of information for a team of students competing in a programming contest. The class has the following public methods:

TeamData(name, problems)	constructs a TeamData object with the given team name and the given number of problems
success(problem, time)	record the successful completion of the given problem with the given time
solved()	returns the total number of problems solved
time()	returns the total time for problems solved
percentCorrect()	returns the percent of problems solved
toString()	returns a String with name, problems solved, total problems, and total time

Each time a team solves a problem, the success method for the team will be called once giving the problem number and the time for that problem. The TeamData object must keep track of the total number of problems solved and the sum of all of the times. It does NOT have to keep track of which problems have been solved. Below is an example of a typical usage:

```
TeamData team1 = new TeamData("UW Red", 8);
team1.success(3, 18);
team1.success(4, 82);
team1.success(6, 130);
System.out.println(team1);
```

The println should produce the following output:

```
UW Red solved 3 of 8 in 230 minutes
```

Your toString method must exactly reproduce this format. After this interaction, the call team1.solved() would return 3, the call team1.time() would return 230, and the call team1.percentCorrect() would return 37.5.

The TeamData class should implement the Comparable interface. Teams that perform better should be considered to be "less" than other teams so that they appear at the beginning of a sorted list. In general, the team that solves the most problems is considered the best, but when there is a tie for the number of problems solved, then total time determines the winner. The team with the lower total time wins in the case of a tie for total problems solved.

You may assume that all values passed to your methods are valid and that the total number of problems is greater than 0.

.

Space for #5

6. Binary Tree Programming. Write a method evenBranches that returns the number of branch nodes in a binary tree that contain even numbers. A branch node is one that has one or two children (i.e., not a leaf). For example, if a variable t stores a reference to the following tree:

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

then the call t.evenBranches() should return 3 because there are three branch nodes with even values (2, 8 and 6). Notice that the leaf nodes with even values are not included (the nodes storing 0 and 4).

Assume that you are writing a public method for a binary tree class defined as follows:

```
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.

Space for #6

7. Binary Tree Programming. Write a method matches that returns a count of the number of nodes in one tree that match nodes in another tree. A match is defined as a pair of nodes that are in the same position in the two trees relative to their overall root and that store the same data. Consider, for example, the following trees.

```
              +---+               |                  +---+
       tree1-> | 3 |              |          tree2-> | 3 |
              +---+               |                  +---+
             /     \              |                 /     \
            /       \             |                /       \
        +---+       +---+         |            +---+       +---+
        | 4 |       | 7 |         |            | 6 |       | 7 |
        +---+       +---+         |            +---+       +---+
       /     \           \        |           /           /     \
      /       \           \       |          /           /       \
  +---+       +---+      +---+    |      +---+        +---+     +---+
  | 0 |       | 9 |      | 2 |    |      | 0 |        | 9 |     | 2 |
  +---+       +---+      +---+    |      +---+        +---+     +---+
                   \              |           \
                    \             |            \
                    +---+         |            +---+
                    | 8 |         |            | 8 |
                    +---+         |            +---+
```

The overall root of the two trees match (both are 3). The nodes at the top of the left subtrees of the overall root do not match (one is 4 and one is 6). The top of the right subtrees of the overall root match (both are 7). The next level of the tree has 2 matches for the nodes storing 0 and 2 (there are two nodes that each store 9 at this level, but they are in different positions relative to the overall root of the tree). The nodes at the lowest level both store 8, but they aren't a match because they are in different positions. Therefore, these two trees have a total of 4 matches. Thus, tree1.matches(tree2) and tree2.matches(tree1) would each return 4.

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

```
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.

Space for #7

8. **Linked Lists.** Write a method called markMultiples that takes an integer n as a parameter and that adds nodes to a linked list of integers to indicate where the multiples of n occur. For example, suppose that a variable list stores the following sequence of values:

```
[6, 5, 8, 13, 12, 3, 5, 2, 0]
```

and we make the following call:

```
list.markMultiples(3);
```

There are four multiples of 3 in this list (6, 12, 3, and 0). Each should be marked by inserting a 0 in front of it:

```
[0, 6, 5, 8, 13, 0, 12, 0, 3, 5, 2, 0, 0]
```

Notice that even the 0 at the end of the original list has a 0 inserted in front of it because it is a multiple of 3. Assume that you are using a linked list that stores integers, as discussed in lecture:

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

    // post: constructs a node with given data and given link
    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

You are writing a method that will become part of the LinkedIntList class. You may not assume that any particular methods are available. You are NOT to change the data field of the existing nodes in the list. You will, however, construct new nodes to be inserted into the list and you will change the links of the list to include these new nodes. Your method should throw an IllegalArgumentException if the value passed to it is less than 1.

Space for #8