(a) Precisely and concisely explain what a data structure is and why we will spend an entire quarter studying data structures.

(b) Circle the bullet in front of each statement that accurately describes your instructor's course policies:

- Having a copy of all or part of another student's source code, just to look at for a little help, is considered cheating.
- You must submit every lab assignment in order to pass this course.
- The final exam for this course will be comprehensive.
- If you miss more than three lectures, your grade will be dropped by half a letter grade.
- All students using a laptop during class must email a copy of their lecture notes to me before the end of the day.
- Lecture and lab attendance are required.
- Your instructor will give you a free lunch if you invite him to lunch.

Do not circle the bullet in front of any statement that incorrectly describes your instructor's course policies.
Feel free to provide an explanation for your answer if you believe the statement to be ambiguous.

Implement the ArrayList.size() and ArrayList.indexOf(Object) methods using the same assumptions that we made in lecture.

(a) Recall that our LinkedList implementation has one attribute: Node head where Node is an inner class with two attributes: E value and Node next. Assume that this inner class has one constructor: Node(E value) and no additional methods. Implement the LinkedList.size() method.

(b) Give the Big-O time complexity for the following method when a LinkedList of size n is passed as an argument. Justify your answer.

```
public static boolean areUnique(List<String> strings) {
  boolean areUnique = true;
  for(int i=0; areUnique && i<strings.size(); ++i) {
    for(int j=0; areUnique && j<strings.size(); ++j) {
      if(i!=j && strings.get(i).equals(strings.get(j)) {
        areUnique = false;
      }
    }
  }
  return areUnique;
}
```

List the methods that are part of a pure stack interface.

Explain how a pure stack interface could be implemented using a LinkedList.

Complete the method below using recursion (no for/while/do-while loops allowed). Hint: Break the problem into three cases: 1) the list is empty, 2) the list is non-empty and the first value matches the target, or 3) the list is non-empty and the first value does not match the target. Use get(0) to return the first element of the list and remove(0) to remove the first element from the list before the recursive call. You may ignore potential NullPointerExceptions.

```
/**
 * Counts the number of times target is found in data
 * @param data The list of values to be searched over
 * @param target The value to be searched for
 * @return The number of times target is found in data
 */
public static int countMatches(List<Integer> data, int target) {
```

Recall that our BinarySearchTree class had an inner, Node, class that contained three attributes: value, lKid, and rKid. Implement the recursive version of BinarySearchTree.toString() that is called by the method below such that the following method returns a string containing all of the elements in the binary search tree listed in order (separated by commas).

```
public String toString() {
    String result = toString(root);
    return "[" + (result.length()>2 ? result.substring(0, result.length()-2) : "") + "]";
}
```

What is a Set? How does it differ from a List?

Implement Node leftRotate(Node subroot) on a binary search tree where each node contains the following attributes value, rKid, lKid, and parent.

(a) Suppose the following Integers are added to a hash table with a capacity of 6. Illustrate the resulting data structure. Note that the hashCode() method for the Integer class just returns the value of the integer.

0, 8, 1, 5, 6, 7

(b) What is the load factor, L, for this hash table?