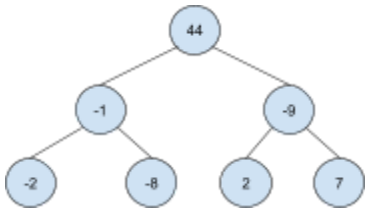
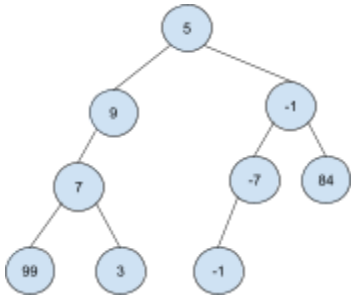
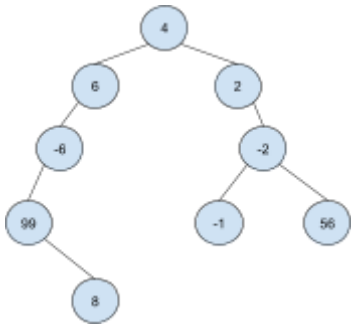
# CSE 123 Winter 2024 Practice Final Exam #2

## 1. Comprehension

**Part A:** For each of the following binary trees, indicate which type of traversal is shown: pre-order, in-order, post-order, or none of these.

| | | |
|---|---|---|
|  | 44 -1 -2 -8 -9 7 2 | ☐ pre-order<br>☐ in-order<br>☐ post-order<br>☐ none |
|  | 99 3 7 9 -1 -7 84 -1 5 | ☐ pre-order<br>☐ in-order<br>☐ post-order<br>☐ none |
|  | 99 8 -6 6 4 2 -1 -2 56 | ☐ pre-order<br>☐ in-order<br>☐ post-order<br>☐ none |

**Part B:** Consider the following method in the `IntTree` class:

```
public void parent() {
      int value = helper(overallRoot);
      if (value == 42) {
            System.out.println("42 is the meaning of life");
      } else {
            System.out.println("Continue the search for purpose…");
      }
}

private int helper(IntTreeNode root) {
      if (root == null) {
            return -2;
      }

      return root.data + helper(root.left) + helper(root.right);
}
```

Draw a binary tree with at least **3 nodes** that, if they were stored in the variable `tree`, the call `tree.mystery()` would print "42 is the meaning of life."

# 2. Code Tracing

Consider the following recursive mystery method:

```
public static boolean mystery(String str) {
	// custom method that removes all spaces found
	// ex: “hello world” -> “helloworld”
	String clean = str.removeAllSpaces();
	return helper(clean);
}

private static boolean helper(String str) {
	if (str.length() <= 1) {
		return true;
	}

	if (str.charAt(0) == str.charAt(str.length() - 1)) {
		return mystery(str.substring(1, str.length() - 1));
	} else {
		return false;
	}
}
```

Assume all string inputs are *lowercase* and have *at least one character*. For each of the following statements, indicate what the output would be:

mystery("rotator") _____

mystery("ufo tofu") _____

mystery("never odd or even") _____

mystery("high noon") _____

mystery("yo banana boy") _____

# 3. Inheritance

Consider the following class:

```
public class RecyclingCenter {
      private String location;
      private Map<String, Integer> materialCounts;
      private double totalWeightProcessed;

      public RecyclingCenter(String location) {
            this.location = location;
            this.materialCounts = new HashMap<>();
            this.totalWeightProcessed = 0;
      }

      public void processMaterials(String materialType, int count,
                                   double weight) {
            materialCounts.put(materialType, count);
            totalWeightProcessed += weight;
      }

      public int getMaterialCount(String materialType) {
            if (matericalCounts.containsKey(materialType))
                  return materialCounts.get(materialType);
            return 0;

      public double getTotalWeightProcessed() {
            return totalWeightProcessed;
      }

      public boolean isEcoFriendly() {
            return totalWeightProcessed >= 10000;
      }
}
```

Write a new class called **AdvancedRecyclingCenter** that represents a more sophisticated recycling center with advanced sorting and processing capabilities. **AdvancedRecyclingCenter** should extend **RecyclingCenter** but differ in the following ways:

- An **AdvancedRecyclingCenter** has the ability to process electronic waste (e-waste) and keeps track of the integer number of e-waste `eWasteCount` and the double weight `eWasteWeightProcessed`. Include getter methods for the fields.
- **AdvancedRecyclingCenter** has a method `processEWaste()` that takes in a new count and new weight to add.
- **AdvancedRecyclingCenter** overrides the `isEcoFriendly()` method. In addition to the base class's criteria, an **AdvancedRecyclingCenter** is also considered eco-friendly if it has processed at least 1000 kilograms of e-waste.

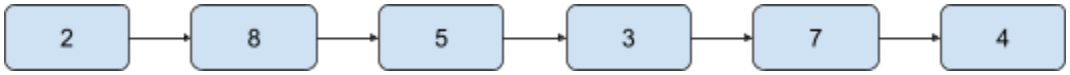
To earn an E on this problem, your **AdvancedRecyclingCenter** class must not duplicate any code from the **RecyclingCenter** class and must not include any unnecessary overrides.

*Write your solution to problem #3 here:*

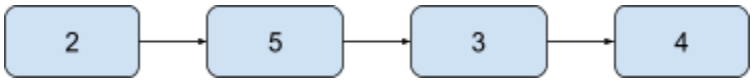
# 4. LinkedList Programming

Write an instance method called **cullMax()** that will be part of the `LinkedIntList` class and accepts an integer value `int val`, and returns a new `LinkedIntList`. All `ListNode` values in the `LinkedIntList` that are less than or equal to the given integer value would stay in the original list while all values greater than the given integer value would be removed from the original and stored in a separate `LinkedIntList`. This list of nodes that have data values greater than the given integer value will be returned.

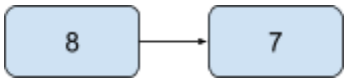
Consider the following given `LinkedIntList` and `val = 5`:




This will be the expected output if we have `LinkedIntList list2 = list1.cullMax(5)`.
Given List (`list1`):




Returned List (`list2`):




To earn an E on this problem, the original order of the provided `LinkedIntList` should be preserved and it should not create any extra objects (other than the new 2nd list). This means auxiliary data structures are NOT allowed. To earn a grade other than an N, the method should still split the `LinkedIntList` in relation to the given integer value.

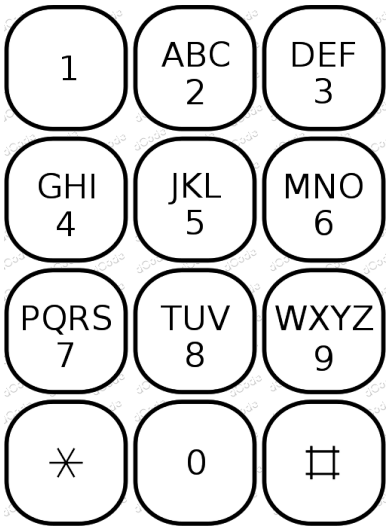
Write your solution on the next page.

*Write your solution to problem #4 here:*

# 5. Recursion Problem

Write a *recursive* method called `letterCombinations` that takes in one parameter, a String `digits` containing digits from 2-9, inclusive. Your method should return all possible letter combinations that a number could represent in a List of Strings. Return the answer in **any order**.

Here's a throwback! A mapping of digits to letters (like on an old-school telephone) is given below. Note that 1 does not map to any letters.




For example, the call `letterCombinations("23")` would produce the following output.

```
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Some other examples include:

```
Input: digits = ""
Output: []

Input: digits = "2"
Output: ["a","b","c"]
```

You may return the combinations in any order, but to earn an E, you must return *all* combinations and the returned list must not contain any combination more than once. To earn a grade other than N, your method **must** be implemented recursively, though you may also use loops as part of your recursive algorithm.

This constant will be provided to you and is encouraged to use in your solution:

```
static String[] KEYPAD = new String[]{"", "", "abc", "def", "ghi", "jkl",
"mno", "pqrs", "tuv", "wxyz"};
```

Write your solution on the next page.

*Write your solution to problem #5 here:*

# 6. Binary Search Tree Problem

Consider the given class:

```
public class DictionaryTree {
	private TreeNode overallRoot;

	public DictionaryTree() {
		overallRoot = null;
	}

	...

	private class TreeNode {
		public String key;
		public int value;
		public TreeNode left, right;

		public TreeNode(String key, int value) {
			this.key = key;
			this.value = value;
			this.left = null;
			this.right = null;
		}
	}
}
```

Add a new method to the given **DictionaryTree** class called **insert()** that accepts a new *key* and *value* to insert into the tree. This method should modify the tree and add a node to the correct spot as described below:

- Compare each key to the current TreeNode:
  - If the *key* is less than the current node's *key*, traverse to the left
  - If the *key* is greater than the current node's *key*, traverse to the right
  - Continue the traversal until a spot is available to insert (i.e. current node is null)
- If the *key* already exists in the Tree, replace the existing *value* with the new given *value*

To earn an E, the node should be inserted into the correct place. Be sure to account for edge cases such as if the tree is empty. To earn a grade other than an N, your method must be implemented recursively, though loops can be used (if necessary) as part of the recursive algorithm.

Write your solution on the next page.

*Write your solution to problem #6 here:*