Lab 2: Arrays and Data Structures

Arrays in Rust

Consider the following code:

```
fn main(){
        let mut groups = [[""; 4]; 6];
        groups[0]=["Bob", "Carol", "Eric", "Matt"];
        groups[1]=["Jim", "Lucy", "Terry", "Brenda"];
        groups[2]=["Susan", "Brad", "Jim", "Matt"];
        groups[3]=["Sue", "Wendy", "Sam", "Brad"];
        groups[4]=["Kate", "Jack", "James", "Sydney"];
        groups[5]=["Mary", "John", "Ricky", "Wendy"];

}
```

This main function contains the names of the members in six research group.

Question 1: You need to write a new function called *searchMember* to search for a member and report the following information:

- Their name's existence in the list (yes or not)
- Their “group number” (some members are in more than one group)
- Whether they are a group leader (the first person listed in each group is indicated as its leader)

DEMO this deliverable to the lab instructor.

Binary Tree

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the *left* child and the *right* child. In other words, each node in a binary tree:

1- must have a value
2- may or may not have left and/or right child

One can describe a single node as follows:

```
#[derive(Debug)]
struct TreeNode {
    data: &str,
    left_child: Option<TreeNode>,
    right_child: Option<TreeNode>,
}
```

Question 2: Try to run the above code. Does it run? If not, explain why and rewrite the code so it runs.

- **DEMO this deliverable to the lab instructor.**

Question 3: Write insert_node function that inserts a node with a given value. You can use the following code snippet.

```
    pub fn insert_node(&self, data: &str) {
        if self.data == data {
            return
        }
        let new_node = if data < self.data { &self.left_child } else {
&self.right_child };
        match ……
    }
```

- **DEMO this deliverable to the lab instructor.**

Question 4: Let's assume your `TreeNode` struct is replaced with the following `Tree` enum:

```
#[derive(Debug)]
enum Tree<T: Ord> {
        Node {
                data: T,
                left_child: Box<Tree<T>>,
                right_child: Box<Tree<T>>,
        },
        Empty,
}
```

What changes do you need to make to your insertion code to run the code?

What is the purpose of Empty?

Which solution (struct-based or enum-based) is better?

- **DEMO this deliverable to the lab instructor.**