Name: _____ Andrew Id: _____

**15-121 Fall 2021 Quiz 7**

Up to 25 minutes. Show your work. No calculators, no notes, no books, no computers, no other people.
Don't write import lines in free response problems.

For all of the questions on this quiz, assume the existence of the following doubly linked list. (It is a simplified version of what was in the homework.) You can assume that `head`, `tail`, and `size` are all properly updated in any of the provided methods below.

```
public class DoubleListNode<NodeDataType> {
    public NodeDataType data;
    public DoubleListNode<NodeDataType> next;
    public DoubleListNode<NodeDataType> prev;

    public DoubleListNode(NodeDataType data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }

    public String toString() {
        return data.toString();
    }
}

public class DoublyLinkedList<ListType> {
    private DoubleListNode<ListType> head;
    private DoubleListNode<ListType> tail;
    private int size;

    public DoublyLinkedList() {
        // You may assume this code exists and works as described in the homework.
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (DoubleListNode<ListType> tmpNode = this.head; tmpNode != null; tmpNode = tmpNode.next) {
            sb.append(tmpNode);
            sb.append(" --> ");
        }
        sb.append("<null>");
        return sb.toString();
    }

    public void addAfter(DoubleListNode<ListType> prevNode, ListType value) {
        // You may assume this code exists and works as described in the homework.
    }

    public void addBefore(DoubleListNode<ListType> postNode, ListType value) {
        // You may assume this code exists and works as described in the homework.
    }

    public ListType remove(DoubleListNode<ListType> node) {
        // You may assume this code exists and works as described in the homework.
    }

    private DoubleListNode<ListType> nodeAtIndex(int index) {
        // You may assume this code exists and works as described in the homework.
    }

    public DoubleListNode<ListType> getHead() {
        return this.head;
    }

}
```

1. (5 points) **Code Tracing**: Assuming that the following two methods are added to the DoublyLinkedList class on the first page of this quiz, what will the output be when `main` is executed?

```
public void mystery(DoubleListNode<ListType> node) {
    DoubleListNode<ListType> t = node.next;
    node.prev.next = t;
    t.prev = node.prev;
    node.next = t.next;
    t.next.prev = node;
    t.next = node;
    node.prev = t;
}

public static void main(String[] args) {
    DoublyLinkedList<Integer> d = new DoublyLinkedList<Integer>();
    d.addBefore(null, 10);
    DoubleListNode<Integer> tmp = d.getHead();
    d.addBefore(tmp, 20);
    d.addBefore(tmp, 30);
    d.addAfter(tmp, 40);
    d.addAfter(tmp, 50);
    System.out.println(d);
    d.mystery(tmp);
    System.out.println(d);
}
```

2. **Free Response**. Write the following methods that go inside of the `DoublyLinkedList` class.

   (a) (5 points)

```
/**
 * Add all the items from an ArrayList to the end of this linked list, in order.
 * This function should run in O(N) time or better.
 *
 * @param arr The ArrayList containing the items to add to the list.
 */
public void addList(ArrayList<ListType> arr) {
```

(b) (5 points)

```
/**
 * Remove half the instances of value in the list.
 *
 * For example, if there are 8 instances of value in the list, then this method
 * will remove the first 4 of them. (But leave the last 4.) If the number of
 * instances of value is odd, then remove half rounded down. (So, if there 7
 * instances, then remove the first 3 of them.)
 * This function should run in O(N) time or better.
 *
 * @param value The value to search the list for.
 */
public void removeHalf(ListType value) {
```

(c) (5 points)

```
/**
 * Return a new DoublyLinkedList containing a "slice" of the data from the original.
 * This function must be non-destructive.
 *
 * For example, if the original list (named bob) contains:
 * 5 --> 6 --> 7 --> 8 --> 9
 * Then bob.slice(1,3) would return a new list containing:
 * 6 --> 7 --> 8
 *
 * For simplicity, you may assume that start and end are valid for the list in
 * question.
 * This function should run in O(N) time or better.
 *
 * @param start The index of the first item to include in the slice.
 * @param end The index of the last item for the slice.
 * @return A new DoublyLinkedList containing only the elements from the slice.
 */
public DoublyLinkedList<ListType> slice(int start, int end) {
```