

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE): **KEY**

STUDENT ID (e.g. 123456789):

INSTRUCTIONS:

Assume the code below with all necessary import statements.

```
public class MyLinkedList<T extends Comparable<T>> {

    private class Node {
        private T data;
        private Node next;
        private Node previous;

        private Node(T data) {
            this.data = data;
            previous = next = null;
        }
    }

    private Node head, tail;

    public MyLinkedList() {
        head = tail = null;
    }

    /* Makes a linear singly linked list with each new item being the new head */
    public MyLinkedList<T> add(T data) {

        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;

        return this;
    }

    public void convertToCDL(){
//you will write this method which will call the private recursive convertToCDLAux
    }

    public String toString() {
        String result = "\\ " ";
        Node curr = head;

        while (curr != null) {
            result += curr.data + " ";

            curr = curr.next;
        }

        return result + "\\ " ";
    }
}
```

```

    public String toStringCircle() {

        String result = "\n ";

        Node curr = tail;

        while (curr !=null){
            result += curr.data + " ";
            curr = curr.previous;
            if (curr==tail) //back to Tail
                break;
        }

        return result + "\n";
    }

}

public class SampleDriver {

    public static void main(String[] args) {

        String answer = "";

        MyLinkedList<String> newList = new MyLinkedList<String>();
        newList.add("Sarah").add("Rose").add("Peter").add("Kelly").add("Albert");
        answer+= newList+"\n";
        newList.convertToCDL();
        answer+= newList.toStringCircle();

        System.out.println(answer);

    }

} }

```

Driver Output

```

" Albert Kelly Peter Rose Sarah "
" Sarah Rose Peter Kelly Albert "

```

A call to `convertToCDL` converts the singly linked list (as seen in class examples and created by repeated calls of the `add` method) to a circular doubly linked list. This just means the `previous` field of each node will point to the previous node, the previous of `head` points to `tail` (since it is a circle), and the `next` field of the `tail` points to `head` instead of `null`. Notice that during the creation of the singly linked list via calls to `add`, the `tail` and `previous` field are not modified, and therefore will contain their default value of `null` when `convertToCDL` is called.

If `convertToCDL` is called on an empty list (`head` is `null`), just return. Otherwise calls the private `convertToCDLAux` to recursively assign the appropriate value to each `previous` field of the nodes. Assigning a value to the `tail` field, the `previous` field of `head`, and the `next` of `tail` can be done when the recursion is done or in `convertToCDL`. You can decide on the number and types of the parameter of `convertToCDLAux` and its return value (`void` is ok). But you cannot change the header of `convertToCDL`:

```

public void convertToCDL(){

```

You cannot have any loops in your code, make any nodes (or new list), or use any auxiliary data structures (array, ArrayList, sets, etc.). Assume that `convertToCDL` is called only once on the singly linked list created by `add`. In other words, don't worry about calling `convertToCDL` on a list that has already been converted.

```
public void convertToCDL(){
    if (head == null) //empty case
    {
        tail = null; //not needed
        return;
    }

    convertToCDLAux(null, head);

    tail.next = head;
    head.previous = tail;
}

private void convertToCDLAux(Node prev, Node headAux) {
    if (headAux != null) {
        convertToCDLAux(headAux, headAux.next); //go to end
        headAux.previous = prev;
    }
    else {
        tail = prev; //set the tail
    }
}

}
```