Problem 2: (tops, spot, ... (18 points))

The method last2first below correctly moves the last node to the front of a linked-list and returns a pointer to the new first node. Thus the call list = last2first(list) would change

("ape", "bat", "cat", "dog")
to the list

("dog", "ape", "bat", "cat")

```java
public static Node last2first(Node list){
    if (list == null || list.next == null) return list;
    // contains at least two nodes
    Node temp = list;
    while (temp.next.next != null) {
        temp = temp.next;
    }
    temp.next.next = list;
    list = temp.next;
    temp.next = null;
    return list;
}
```

Part A (3 points)
What is the big-Oh complexity of last2first when executed on an N-node list? Justify your answer briefly.

Part B (3 points)
Describe the contents of the linked-list referenced by list after the code fragment below executes when list is an n-node list.

```java
for(int k=0; k < n; k++){
    list = last2first(list);
}
```
Part C: Star Trap (4 points)
The method `stuff` below returns a one-node list unaltered. If `list` represents ("ape", "cat"), what is the result of executing

```java
list = stuff(list);
```

and what is the result of executing the statement when `list` is ("ape", "cat", "moose")?

```java
class Node {
    String value;
    Node next;
}
public static Node stuff(Node list) {
    if (list == null || list.next == null) return list;
    // contains at least two nodes
    list = last2first(list);
    list.next = stuff(list.next);
    return list;
}
```

(you should have two answers to the above question).

Part D: Again Please (8 points)
Describe the runtime complexity using big-Oh of executing `stuff` with an N-node list and describe the list returned (its values and their order compared to the order before the method executes). Justify your answers.
PROBLEM 3 : \textit{(ILST (16 points))}

Part A (8 points)

In this problem assume that the Node class has a field prev and that lists are doubly-linked. When a list is sorted it is possible to determine the number of unique/different elements in $O(n)$ time for an $n$-node sorted, doubly-linked list. For example there are 4 unique values in the list

("one","one","one","roo","star","star","tree","tree","tree")

Write method unique that returns the number of unique values in a sorted, doubly-linked list. You may not use any sets, maps, arraylists, etc. For full credit your routine should run in $O(n)$ time for an $n$-element list.

```java
/**
 * @param list is sorted, doubly linked
 * @return the numbr of different/unique values in list
 */
public static int unique(Node list){
```
Part B (8 points)
Write method `alternatingList` that returns a linked list constructed from the values in an array. The linked list contains the same values in the array but the values with even indexes appear before the values with odd indexes (the order of the values with even indexes is the same as the order in the array, and the same for odd indexes). For example, for the array `[a,b,c,d,e,f,g]` The list below would be returned (not doubly linked).

```java
public static Node alternatingList(String[] list){
    Node first = new Node(list[0],null);
    return first;
}
```
PROBLEM 4:  \((LinkedIn \ (10 \ points))\)

Using the \texttt{Node} class at the beginning of the test the code below prints the string values in a linked list on one line.

```java
public void print(Node list) {
    while (list != null) {
        System.out.print(list.info + " ");
        list = list.next;
    }
    System.out.println();
}
```

You'll write the method \texttt{twin} described below that duplicates each node of a linked list. The code below should generate the output beneath it when \texttt{twin} is written.

```java
public void linkTest() {
    print(list);
    list = twin(list);
    print(list);
}
```

Output:

```
child dog cat spit
child child dog dog cat cat spit spit
```

Part A (5 points)

Given a list of \(n\) nodes the code you write in \texttt{twin} should create \(n\) new nodes and return a pointer to the first node of a list containing \(2n\) nodes as described above. Be sure you assign values to the \texttt{next} fields of nodes in linking new nodes into the list.

```java
public Node twin(Node list) {
```

(continued)
Part B (5 points)
Write the method `removeOther` that removes every other node from a linked-list and returns a pointer to the first node of the new list. For example the code below prints the original list, i.e.,

```
child dog cat spit
```

```java
class Node {  
    String data;  
    Node next;  
    public Node(String data, Node next) {this.data = data; this.next = next;}  
    @Override  
    public String toString() {return data;}  
}
```

```java
public void linkTest(){  
    list = twin(list);  
    list = removeOther(list);  
    print(list);  
}
```

To remove a node from a linked list you’ll need to use a line of code similar to the one below which links a node “around” the node after it. What you write should include code like this to change links so that nodes are removed from the list.

```
list.next = list.next.next;
```

Conceptually, if the nodes of a list are numbered 1, 2, 3, … n your code removes all the nodes whose numbers are even, leaving only the odd numbered nodes from the original list. To remove a node you link around it so that nothing in the list points to it.

```java
public Node removeOther(Node list) {
```

```java
}
```