Midterm 2: CompSci 201
Form A

Profs. Astrachan and Nemecek

October 29, 2025

Name: ____

NetID: ____

In submitting this test, I affirm that I have followed the Duke Community Standard.

Community standard acknowledgement (signature) ____

If you use helper sheets, you must put your name and NetID on them.
Some questions have two only two answers to choose from, some have three, some have four. Make sure you bubble answers appropriately.

Below is the definition of `ListNode`, unless another is specified. This is the same definition as used in linked-list APT problems.

```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int x) {
        info = x;
    }
    ListNode(int x, ListNode node) {
        info = x;
        next = node;
    }
}
```

This Exam is Form A. Please make sure your answer sheet is marked accordingly.

The next two problems concern the method `create` shown below. Note that `create("duke",5).length()` is 20.

In class we discussed that the *runtime* of `create("duke",n)` is $O(n^2)$ because Strings are immutable.

```
public String create(String base, int n) {
    String ret = "";
    for(int k=0; k < n; k++){
        ret += base;
    }
    return ret;
}
```

PROBLEM 1:

Using O-notation, what is the *length* of the the String *returned* by the call `create("duke",n)`?

A. $O(n)$

B. $O(n^2)$

C. $O(n^3)$

D. $O(n^4)$

PROBLEM 2:

Using O-notation, what is the *runtime* of executing `create(create("duke",n),n)`? Note that `create(create("duke",2),2)` returns a String of length 16.

A. $O(n)$

B. $O(n^2)$

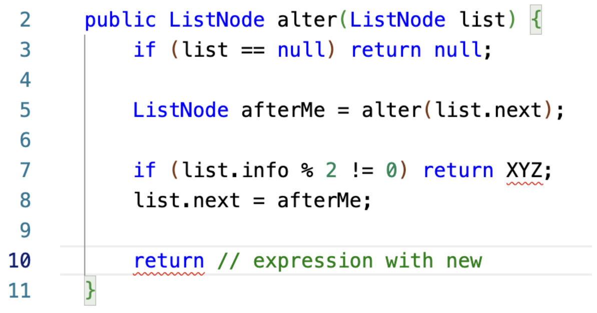
C. $O(n^3)$

D. $O(n^4)$

Several problems deal with a previous APT (not part of those you've been given) that required writing a method `alter` that removed all nodes containing an odd number, and duplicated all nodes containing an even number.

For example, if `list->1->2->3->4` then `alter(list)` returns a pointer: `list->2->2->4->4`. As another example, if `list->6->4->3` then `alter(list)` returns `list->6->6->4->4`. If all nodes in `list` contain odd values, then `alter(list)` returns `null`.

The code below is all green when lines 7 and 10 are completed correctly.



```
 2  public ListNode alter(ListNode list) {
 3      if (list == null) return null;
 4
 5      ListNode afterMe = alter(list.next);
 6
 7      if (list.info % 2 != 0) return XYZ;
 8      list.next = afterMe;
 9
10      return // expression with new
11  }
```

PROBLEM 3:

The statement `return XYZ` on line 7 deals with the case that `list.info` is odd, (so that `list.info % 2 != 0`) What value replaces `XYZ` in an all-green solution?

A. `list`

B. `null`

C. `list.next`

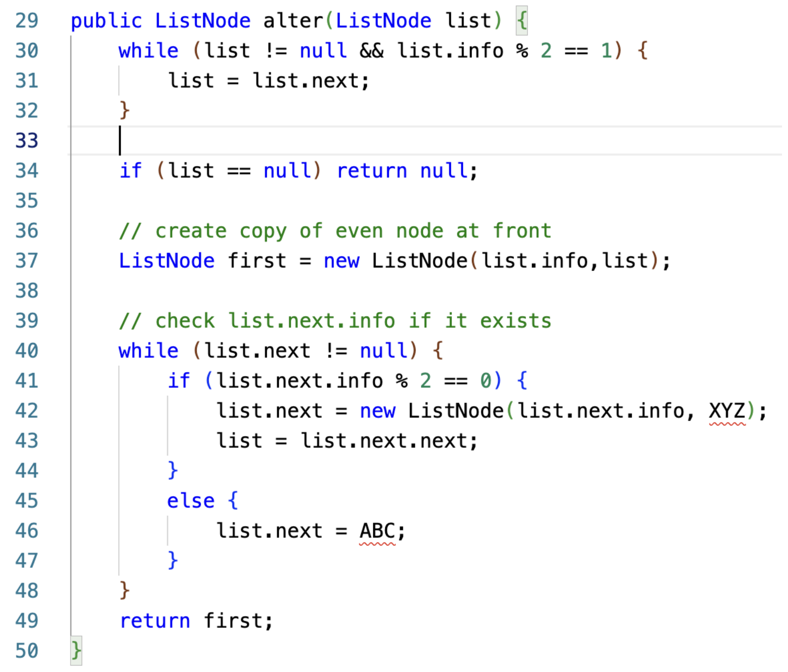
D. `afterMe`

PROBLEM 4:

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

The `return` statement on line 10 indicates that the expression returned should call `new`. The call is `return new ABC`, where `ABC` is replaced by a call to a `ListNode` constructor. What is the call of the `ListNode` constructor that leads to an all-green solution?

Consider an iterative solution as shown in the code below. The code finds the first node containing an even value (if such a node exists), duplicates that node, then iterates through the list removing odd nodes and duplicating even nodes.



```
29 public ListNode alter(ListNode list) {
30     while (list != null && list.info % 2 == 1) {
31         list = list.next;
32     }
33     
34     if (list == null) return null;
35 
36     // create copy of even node at front
37     ListNode first = new ListNode(list.info,list);
38 
39     // check list.next.info if it exists
40     while (list.next != null) {
41         if (list.next.info % 2 == 0) {
42             list.next = new ListNode(list.next.info, XYZ);
43             list = list.next.next;
44         }
45         else {
46             list.next = ABC;
47         }
48     }
49     return first;
50 }
```

PROBLEM 5:

Which of the following best explains the purpose of the loop on lines 30-32?

A. It advances `list` so that the value of `list` will be a pointer to the first even node, or be null.

B. It advances `list` so that `list.next == null`, the last node.

C. It simulates the base case in a non-recursive implementation of `alter`.

PROBLEM 6:

The code on line 42 duplicates `list.next` if that node contains an even value. For example, if `list->X->4->3->...`, then `list.next == 4` and the code one line 42 adds a node so `list->X->4->4->3->...` (Note that `list` points to `X` and `list.next` points at 4.)

The first node containing 4 is the node created when `new` is called. Note that line 43 then advances `list` so that it points at the second 4, which is the original node containing 4 (not the new node).

What is the value of `XYZ` so the code works as intended?

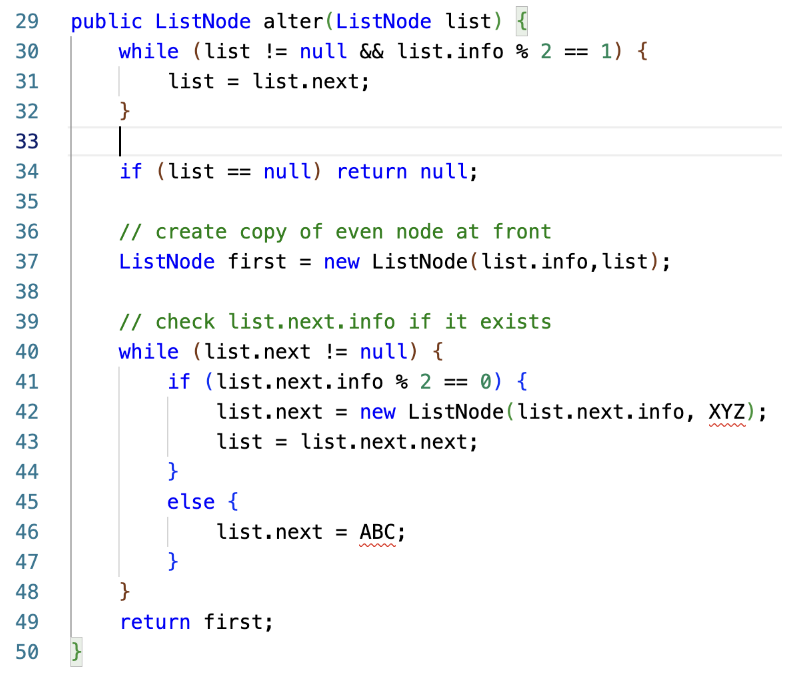
A. `null`

B. `list`

C. `list.next`

D. `list.next.next`

(code duplicated)



```java
29 public ListNode alter(ListNode list) {
30     while (list != null && list.info % 2 == 1) {
31         list = list.next;
32     }
33 
34     if (list == null) return null;
35 
36     // create copy of even node at front
37     ListNode first = new ListNode(list.info,list);
38 
39     // check list.next.info if it exists
40     while (list.next != null) {
41         if (list.next.info % 2 == 0) {
42             list.next = new ListNode(list.next.info, XYZ);
43             list = list.next.next;
44         }
45         else {
46             list.next = ABC;
47         }
48     }
49     return first;
50 }
```

PROBLEM 7:

Suppose that when line 41 is executed, `list->X->4->Y->...` so that `list.next.info == 4` (note that `list` points to `X` and `list.next` points at 4). *After line 43 executes* what is the value in the node pointed to by `list`?

A. `X`

B. `4`

C. `Y`

D. it cannot be uniquely determined

PROBLEM 8:

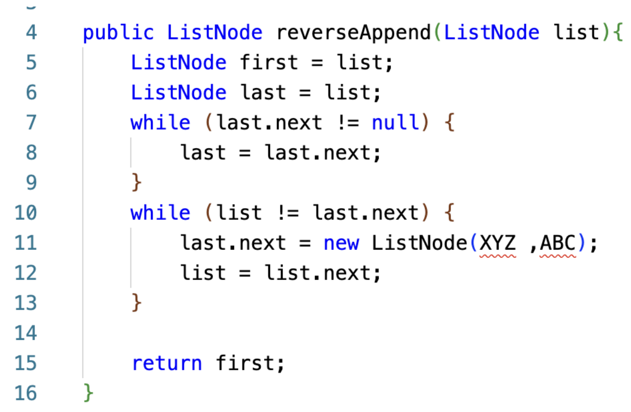
The code on line 46 is intended to link around a node containing an odd value so that it is removed from the list. If `list->X->3->Y->...` then this line causes `list->X->Y->...` (Note that `list` points to `X` and `list.next` points at 3.)

What is the value of `ABC` to make the code all-green?

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

The next four problems are about a method `reverseAppend` that adds N new nodes to the end of a list of N nodes, returning a list of $2N$ nodes. For a list `1->2->3->4`, the call `reverseAppend(list)` returns the list `1->2->3->4->4->3->2->1` where the last four nodes are new and the first four are the original nodes. In general, the last n nodes in the returned list are new nodes that contain the reverse of the original first n nodes.

The code below is all-green when line 11 is completed correctly.



```
4   public ListNode reverseAppend(ListNode list){
5       ListNode first = list;
6       ListNode last = list;
7       while (last.next != null) {
8           last = last.next;
9       }
10      while (list != last.next) {
11          last.next = new ListNode(XYZ ,ABC);
12          list = list.next;
13      }
14
15      return first;
16  }
```

PROBLEM 9:

What value is returned for the call `reverseAppend(null)`, i.e., when an empty list is passed as a parameter?

A. The empty list, since reversing and appending nothing yields an empty list

B. A list containing one new node containing the value 1

C. Nothing is returned, a null-pointer exception is thrown when line 7 executes.

PROBLEM 10:

The code works as intended when `XYZ` on line 11 is replaced by an integer value. What is that value?

A. `list.info`

B. `first.info`

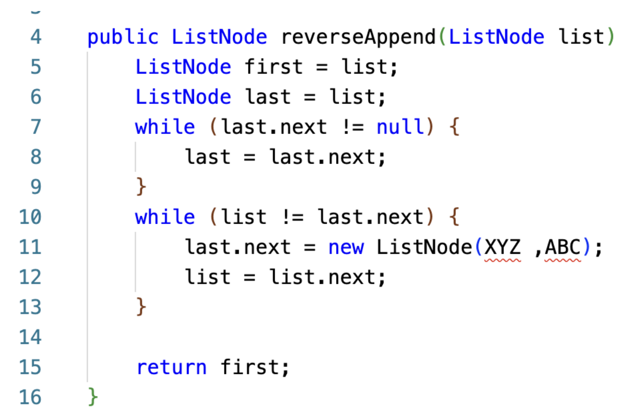
C. `last.info`

PROBLEM 11:

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

What is the value of expression `ABC` on line 11 that makes the code correct?

(code duplicated)



```java
public ListNode reverseAppend(ListNode list){
    ListNode first = list;
    ListNode last = list;
    while (last.next != null) {
        last = last.next;
    }
    while (list != last.next) {
        last.next = new ListNode(XYZ ,ABC);
        list = list.next;
    }

    return first;
}
```

PROBLEM 12:

If `list` contains N nodes, the runtime of the two calls:

`reverseAppend(list);` `reverseAppend(reverseAppend(list));`

are:

A. The same, both are $O(N)$.

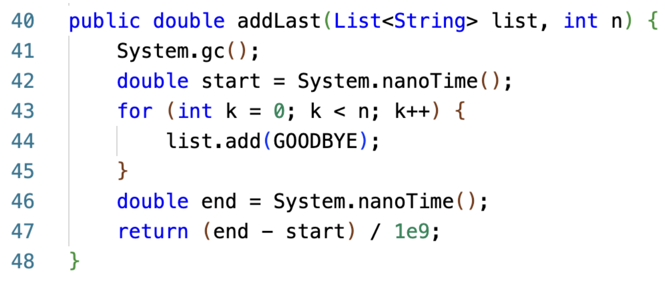
B. The same, both are $O(N^2)$.

C. Different, the runtime of `reverseAppend(list)` is $O(N)$ whereas the runtime of `reverseAppend(reverseAppend(list))` is $O(N^2)$.

D. Different, the runtime of `reverseAppend(list)` is $O(N^2)$ whereas the runtime of `reverseAppend(reverseAppend(list))` is $O(N)$.

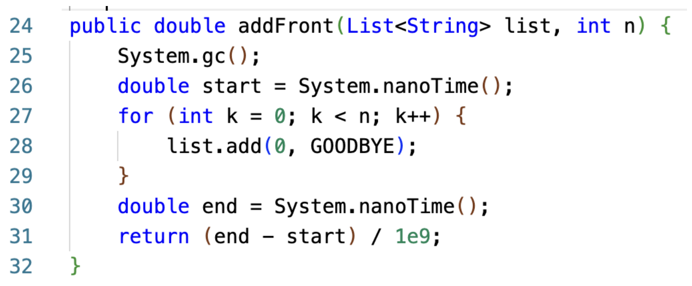
The next four problems concern different methods to which both a `LinkedList<String>` object and an `ArrayList<String>` object are passed. The first two are methods we saw in class: `addLast`, which adds n elements to the end of a `List<String>`, and `addFront`, which adds n elements to the front of a `List<String>`.

We saw in class that for the method `addLast` below, the calls `addLast(ArrayList<String>)` and `addLast(LinkedList<String>)` are both $O(N)$ for N-element lists.



```java
public double addLast(List<String> list, int n) {
    System.gc();
    double start = System.nanoTime();
    for (int k = 0; k < n; k++) {
        list.add(GOODBYE);
    }
    double end = System.nanoTime();
    return (end - start) / 1e9;
}
```

We also saw that for the method `addFront` below, the call `addFront(ArrayList<String>)` is $O(N^2)$ whereas `addLast(LinkedList<String>)` is $O(N)$ for N-element lists.



```java
public double addFront(List<String> list, int n) {
    System.gc();
    double start = System.nanoTime();
    for (int k = 0; k < n; k++) {
        list.add(0, GOODBYE);
    }
    double end = System.nanoTime();
    return (end - start) / 1e9;
}
```

PROBLEM 13:

Which of the following is the best explanation for why `addFront(ArrayList<String>)` is $O(N^2)$?

A. ArrayLists must grow an internal array by doubling it in size when that array is full.

B. Adding an element at the front requires shifting other elements in the array and the sum of the numbers from 1 to N is $N \cdot (N+1)/2$.

C. Garbage collection with `ArrayList` objects take longer than with `LinkedList` objects because of contiguous memory.

PROBLEM 14:

Which is the best explanation as to why it is possible to pass both an `ArrayList<String>` and a `LinkedList<String>` to these methods whose parameter has type `List<String>`?

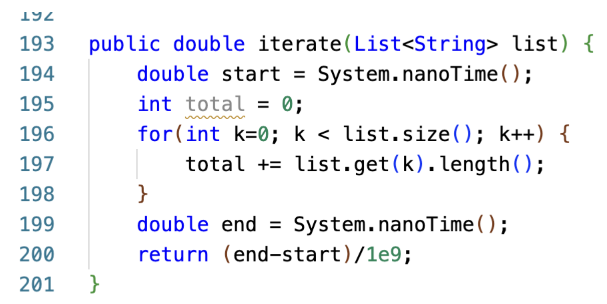
A. There are actually two versions of each method, one for `ArrayList` and one for `LinkedList`, that are generated by the Java compiler.

B. Both `ArrayList` and `LinkedList` implement the `List` interface, and the methods are written using the method `.add` which is in the `List` interface.

C. Java is an object-oriented language and the class `ArrayList` and `LinkedList` both extend the class `Object`.

PROBLEM 15:

Consider the method `iterate` shown below that calculates the total of the lengths of every String in parameter `list`. The runtime of the method depends on the runtime of `list.get(k)` for `ArrayList` and `LinkedList`.



```java
public double iterate(List<String> list) {
    double start = System.nanoTime();
    int total = 0;
    for(int k=0; k < list.size(); k++) {
        total += list.get(k).length();
    }
    double end = System.nanoTime();
    return (end-start)/1e9;
}
```

Which one of the following is correct?

A. The runtime is $O(N)$ for `ArrayList` and $O(N^2)$ for `LinkedList` parameters.

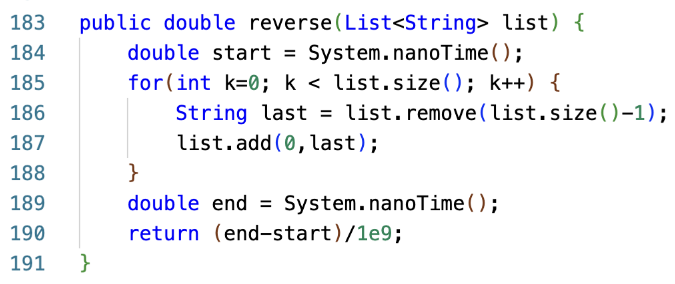
B. The runtime is $O(N)$ for `LinkedList` and $O(N^2)$ for `ArrayList` parameters.

C. The runtime is $O(N)$ for both `LinkedList` and `ArrayList` parameters.

D. The runtime is $O(N^2)$ for both `LinkedList` and `ArrayList` parameters.

PROBLEM 16:

Consider the method `reverse` shown below that reverses its parameter `list`, e.g., if `list = [a,b,c,d]`, the call `reverse(list)` changes the parameter so `list = [d,c,b,a]`.



```java
public double reverse(List<String> list) {
    double start = System.nanoTime();
    for(int k=0; k < list.size(); k++) {
        String last = list.remove(list.size()-1);
        list.add(0,last);
    }
    double end = System.nanoTime();
    return (end-start)/1e9;
}
```

Which of the following is correct?

A. The runtime is $O(N)$ for `ArrayList` and $O(N^2)$ for `LinkedList` parameters.

B. The runtime is $O(N)$ for `LinkedList` and $O(N^2)$ for `ArrayList` parameters.

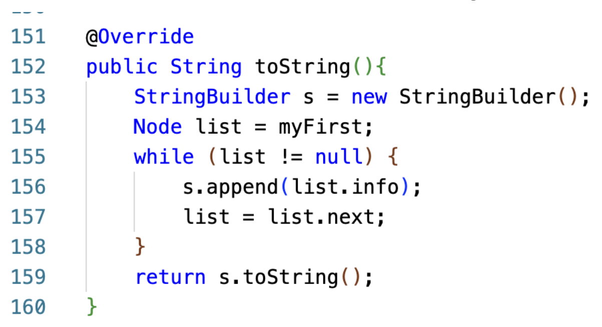
C. The runtime is $O(N)$ for both `LinkedList` and `ArrayList` parameters.

D. The runtime is $O(N^2)$ for both `LinkedList` and `ArrayList` parameters.

The next questions concern the P3: DNA project.

PROBLEM 17:

A correct verion of `LinkStrand.toString` from Project P3 is shown below.



```java
@Override
public String toString(){
    StringBuilder s = new StringBuilder();
    Node list = myFirst;
    while (list != null) {
        s.append(list.info);
        list = list.next;
    }
    return s.toString();
}
```

If `StringBuilder` is replaced by `String`, and `append` is replaced by `concat`, this code still *works correctly*. Which one of the following is true regarding this new version of the `toString` method that uses `String` rather than `StringBuilder`? Note that `s.concat(t)` is the same as `s = s + t` for `String` objects.

A. Both versions are $O(N)$ for a strand with N characters (one per node).

B. The version with `String` is $O(N^2)$ where as the version with `StringStrand` is $O(N)$ for a strand with N characters (one per node).

C. The version with `StringBuilder` is $O(N^2)$ where as the version with `String` is $O(N)$ for a strand with N characters (one per node).

PROBLEM 18:

Which one of the following is true about `LinkStrand` compared to the other strand types in saving memory when the method `cutAndSplice` executes, when a `splicee` is large. For example, suppose `splicee` has a million characters (S) and there are 1,000 breaks (b), so that `splicee` appears 1,000 times in the resulting `LinkStrand` strand?

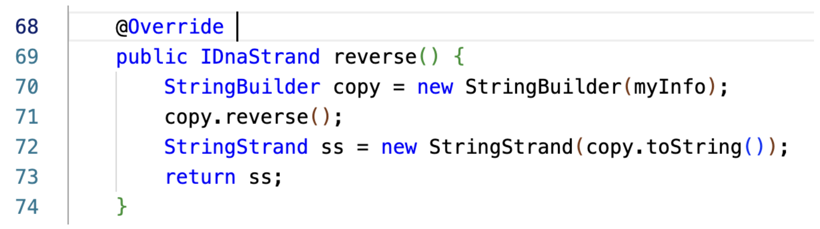
A. Each node created for the 1,000 breaks in the `LinkStrand` points to the same `splicee`.

B. Each node created for the 1,000 breaks in the `LinkStrand` points to a `StringBuilder` copy of `splicee`.

C. Each node created for the 1,000 breaks is part of a circularly linked list so that `myLast` points at the first node and `myFirst` is **not needed**.

PROBLEM 19:

The implementation of `StringStrand.reverse` is shown below (from the code that is part of the project).



```
@Override
public IDnaStrand reverse() {
    StringBuilder copy = new StringBuilder(myInfo);
    copy.reverse();
    StringStrand ss = new StringStrand(copy.toString());
    return ss;
}
```

For a `StringStrand` object `ss` with `ss.size() == N`, what the runtime complexity of the call `ss.reverse()`?

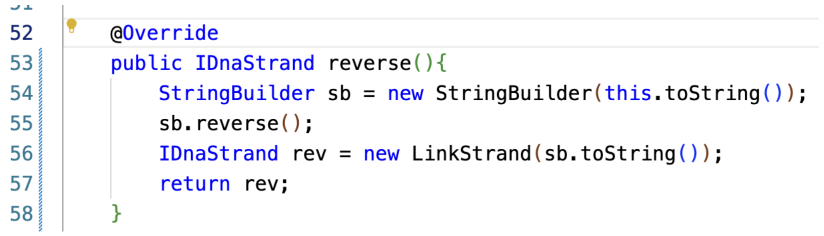
A. $O(1)$

B. $O(N)$

C. $O(N^2)$

PROBLEM 20:

A student proposes the following implementation of `LinkStrand.reverse`, which passes the JUnit tests in the `TestStrand` class.



```
@Override
public IDnaStrand reverse(){
    StringBuilder sb = new StringBuilder(this.toString());
    sb.reverse();
    IDnaStrand rev = new LinkStrand(sb.toString());
    return rev;
}
```

However, this implementation does **NOT** satisfy the requirements in the assignment. For a `LinkStrand` object `link`, all but one of the following statements is true about the `LinkStrand` object returned by the call `link.reverse()`. Which one statement is **NOT** always true using this version of `reverse`?

A. `link.size() == link.reverse().size()`

B. `link.appendCount() == link.reverse().appendCount()`

C. `link.toString().length() == link.reverse().toString().length()`

PROBLEM 21: Consider the method `fun` below. Note that the value returned by `fun(1)` is 0, the value returned by `fun(2)` is 1, and the value returned by `fun(4)` is 2.

```
public int fun(int x) {
    if (x < 2) return 0;
    return 1 + fun(x/2);
}
```

What is the value of `fun(1024)`?

A. $(1024 \cdot 1025)/2$

B. 1

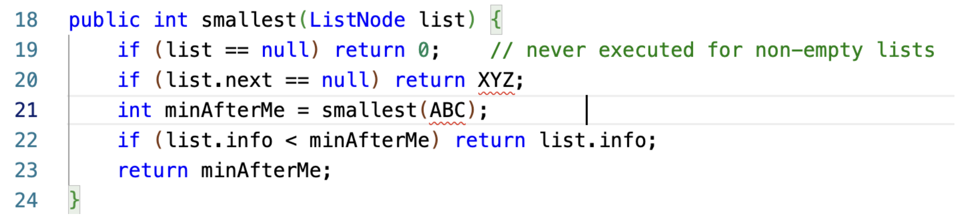
C. 10

D. 100

E. 1024!

For the next two problems, consider the method `smallest` intended to return the minimal value in a non-empty linked-list of `ListNode` objects.

The method is all-green when lines 20 and 21 are completed correctly.



```
18 public int smallest(ListNode list) {
19     if (list == null) return 0;    // never executed for non-empty lists
20     if (list.next == null) return XYZ;
21     int minAfterMe = smallest(ABC);
22     if (list.info < minAfterMe) return list.info;
23     return minAfterMe;
24 }
```

PROBLEM 22:

What value should be returned on line 20 for the base case of a one-node list in place of `XYZ` ?

A. `0`

B. `Integer.MAX_VALUE`

C. `list.info`

PROBLEM 23:

What value should be passed in the recursive call in place of `ABC` on line 21?

A. `null`

B. `list.next`

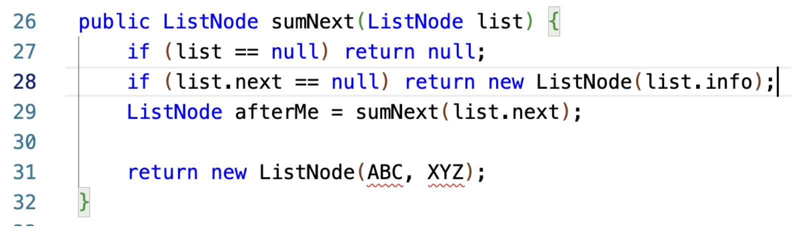
C. `list.next.next`

The method `sumNext` below is intended to return a *new linked-list, leaving the parameter unchanged.*

For example, for `list->1->2->3->4` the call `sumNext(list)` returns `3->5->7->4` (leaving `list` unchanged).

More generally, each node in the returned list is the sum of a node and its next node in parameter `list`. Since the last node does not have a next node, its value appears in the returned list unchanged.

As another example, if `list->4->2->3->8` the call `sumNext(list)` should return `6->5->11->8` (since 4+2 = 6, 2+3 = 5, and 3+8 = 11).



```
26  public ListNode sumNext(ListNode list) {
27      if (list == null) return null;
28      if (list.next == null) return new ListNode(list.info);
29      ListNode afterMe = sumNext(list.next);
30  
31      return new ListNode(ABC, XYZ);
32  }
```

PROBLEM 24:

If the return statement on line 28 is replaced by `return null` and line 31 is correct, how does the returned list change?

A. The returned list would have one less node than in the original code.

B. The returned list would be the same as in the original code, there is no change.

C. The returned list would now be circular.

PROBLEM 25:

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

What is the value of `ABC` on line 31 for the method to be correct?

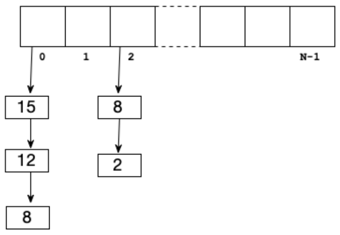
PROBLEM 26:

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

What is the value of `XYZ` on line 31 for the method to be correct?

Several problems deal with this scenario:

Consider an array of linked-lists, for example `ListNode[] array` where each element of `array` is a pointer to a linked list as diagrammed below, so that `array[0]` points to the first node of the list [15,12,8]; `array[2]` points to the first node of the list [8,2]; and other array values point at other lists not shown.



PROBLEM 27:

The method `sumArray` is designed to return the sum of all the linked lists that are referenced in parameter `array` as diagrammed above.

```
28  public int sumArray(ListNode[] array) {
29      int total = 0;
30      for(ListNode list : array) {
31          // statement
32      }
33      return total;
34  }
```

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

What is the statement on line 31 that completes this method? That statement must call the method `sum` shown below.

```
public int sum(ListNode list) {
   if (list == null) return 0;
   return list.info + sum(list.next);
}
```

PROBLEM 28:

Method `find`, partially completed below, should return a pointer to the node of a linked list containing `target`, or `null` if no node in the linked list contains `target`. For example, the call `find(array[0],8)` returns a pointer to the node containing 8 given the array diagrammed above; the call `find(array[2],5)` returns `null`.

```
41  public ListNode find(ListNode list, int target) {
42      if (list == null) return null;
43      if (list.info == target) return ABC;
44      return find(XYZ,target);
45  }
```

What replaces `ABC` on line 43 for the method to work as intended?

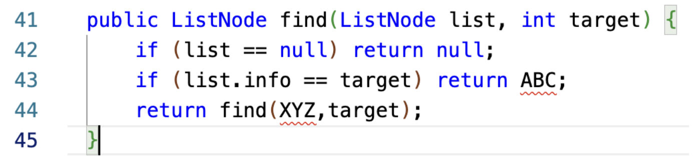
A. `list`

B. `list.next`

C. `find(list.next,target)`

PROBLEM 29:

(code reproduced)



```
41  public ListNode find(ListNode list, int target) {
42      if (list == null) return null;
43      if (list.info == target) return ABC;
44      return find(XYZ,target);
45  }
```

What replaces `XYZ` on line 44 for the method to work as intended?

A. `list`

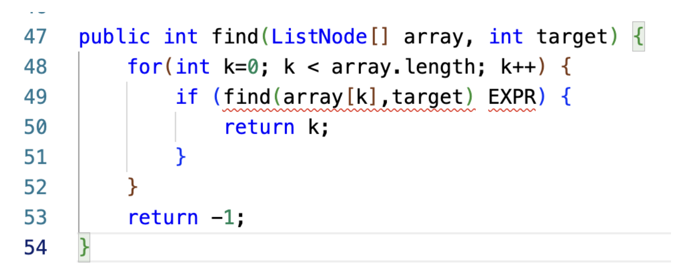
B. `list.next`

C. `list.next.next`

PROBLEM 30:

A different method named `find` is intended to return the index of the linked list in `array` that contains `target`, or -1 if there is no linked list in `array` that contains `target`. If more than one linked list in `array` contains `target`, the smallest index is returned.

For example, in the list diagrammed above `find(array,2)` returns 2, `find(array,12)` returns 0, and `find(array,8)` returns 0, since 0 is less than 2. **Note that `find` on line 49 refers to the method in the previous two problems where the first parameter is a `ListNode`.**



```
47  public int find(ListNode[] array, int target) {
48      for(int k=0; k < array.length; k++) {
49          if (find(array[k],target) EXPR) {
50              return k;
51          }
52      }
53      return -1;
54  }
```

Which of the following replaces `EXPR` for the method to work as intended?

A. `!= null`

B. `== null`

C. `< array.length`

D. `<= array.length`