True/False question:

Version A: F, T, F, T, F, T, F, F, F, F

Version B: T, F, F, T, F, F, T, F, F, F

Dually linked list question (Q3 in A and Q2 in B):

```
public boolean repOk(){
        //the list has at least one node
        if(header == null){
                return false;
        }
        Node current = header;
        Node current2 = header;
        Set<Node> visited = new HashSet<Node>();
        while(current != null){
                //next1 and next2 points to the same node or they are both null
                if(current.next1 != current.next2){
                        return false;
                }

                //the list has no cycle
                if(!visited.add(current)){
                        return false;
                }

                current = current.next1;
                current2 = current.next2;
        }

        //in case current is null and current2 points to a non-null reference
        if(current != current2){
                return false;
        }
        return true;
}
```

Grammar question (Q5 in A and Q3 in B):

(a) Zero.zero;

(b) new Minus(new Plus(Zero.zero, One.one), Two.two);

(c) new Plus(Zero.zero, new Minus(One.one, Two.two));

(d) 21

“0”, ”1”, “2”

“0+0”, “0+1”, “0+2”, “1+0”, “1+1”, “1+2”, “2+0”, “2+1”, “2+2”

“0-0”, “0-1”, “0-2”, “1-0”, “1-1”, “1-2”, “2-0”, “2-1”, “2-2”

or 3 + 2*3*3

(e) “00+” or “+00”

(f)
```
public class NegExpr extends Expression {
        Expression exp;

        public NegExpr(Expression exp) {
                exp = exp;
        }

        public String toString() {
                return “-” + exp;    // No “()”
        }
}
```

Logical coverage question (Q4 in A and Q4 in B):

(a) b = False, c = True

b = False, c = False

(b) a = True, c = True

a = False, c = False

(c) 2 possible ways:

row 3 (T, F, T), row 7 (F, F, T)

row 4 (T, F, F), row 8 (F, F, F)

Input space partitioning question (Q2 in A and Q5 in B):

(a) size of arr. (Or anything that is reasonable)

(b) (Any partitioning that is reasonable)

size of arr is 0	size of arr is 1	size of arr is greater than 1
new String[0]	new String[]{“a”}	new String[] {“a”, “b”}

(c) length of v. (Or anything that is reasonable)

(d) (Any partitioning that is reasonable)

length of v is 0	length of v is 1	length of v is greater than 1
“”	“a”	“ab”

(e) 9 tests in total:

```
assertEquals(count(new String[0], “”), 0);
assertEquals(count(new String[0], “a”), 0);
assertEquals(count(new String[0], “ab”), 0);
assertEquals(count(new String[]{“a”}, “”), 0);
assertEquals(count(new String[]{“a”}, “a”), 1);
assertEquals(count(new String[]{“a”}, “ab”), 0);
assertEquals(count(new String[]{“a”, “b”}, “”), 0);
assertEquals(count(new String[]{“a”, “b”}, “a”), 1);
assertEquals(count(new String[]{“a”, “b”}, “ab”), 0);
```