1 Looping

What output is produced by the following non standard for loop? Record it exactly. We strongly suggest that you use a table to keep track of the value of k.

```java
for (int k = 1; k <= 10; k++) {
    k = k + 1;
    System.out.print(k + " ");
}
```

Output:

2 While to For

Translate each of the following two while loops into for loops.

*Note: You must get both parts correct to get credit for this question.*

Part One

```java
int k = 0;
while (k < 10) {
    System.out.println(k);
    k = k + 1;
}
```

Select the letter corresponding to the for loop below that is equivalent to the while loop above.

A

```java
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

B

```java
for (int i = 0; i < 10; i++) {
    i = i + 1;
    System.out.println(i);
}
```

C

```java
for (int i = 1; i < 10; i++) {
    System.out.println(i);
}
```

D
Loops and Arrays

```java
for (int i = 1; i < 10; i++) {
    i = i + 1;
    System.out.println(i);
}
```

Part Two

```java
int k = 0;
while (k < 10) {
    k = k + 1;
    System.out.println(k);
}
```

Select the letter corresponding to the for loop below that is equivalent to the above while loop.

A
```java
for (int i = 0; i <= 10; i++) {
    System.out.println(i);
}
```

B
```java
for (int i = 0; i <= 10; i++) {
    i = i + 1;
    System.out.println(i);
}
```

C
```java
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

D
```java
for (int i = 1; i <= 10; i++) {
    i = i + 1;
    System.out.println(i);
}
```
3 Arrays

Refer to the following classes:

```java
public class Point {
    public int x;
    public int y;
    //implementation
}

public class Line {
    public Point left_endpoint;
    public Point right_endpoint;
    public int slope;
    public Line(Point one, Point two) {
        this.left_endpoint = one;
        this.right_endpoint = two;
    }
    //implementation
}
```

What is printed by the program below? Record it exactly.

```java
public class Test {
    public static void main(String[] ars) {
        Line[] line = new Line[2];
        Point p = new Point();
        Point q = new Point();
        Line pq = new Line(p, q);
        line[0] = pq;
        pq = new Line(q, p);
        line[1] = pq;
        line[0].left_endpoint.x = 1;
        line[1].left_endpoint.y = 2;
        line[1] = line[0];
        line[0] = pq;
        line[0].left_endpoint.x = 2;
        line[0].left_endpoint.y = 1;
        System.out.print(line[0].right_endpoint.x + " ");
        System.out.print(line[0].left_endpoint.x + " ");
        System.out.print(line[0].right_endpoint.y + " ");
        System.out.print(line[0].left_endpoint.y);
    }
}
```

Output:
4 Build Code

Part One

Note: You must get both parts correct to get credit for this subpart.

You are writing a program `boolean isPrime(int n)` which is designed to determine if a number is prime. You know that the definition of a prime number is one that is divisible only by 1 and itself. Assume that `n` is a positive number and is greater than 1 throughout this entire question.

Which of the following boolean conditions is true for any number `1 < x < n` if the number `n` is prime.

1. `n % x == 1`
2. `n % x == x`
3. `n % x != 0`
4. None of the above

Which of the following would provide an inefficient but accurate implementation of the `boolean isPrime(int n)` method. Note that at least one of the below is correct.

Select the letter corresponding to the correct answer(s).

A

```java
int x = 2;
while (x < n) {
    if (!condition) {
        return false;
    }
}
return true;
```

B

```java
for (int x = 2; x < n; x++) {
    if (!condition) {
        return false;
    }
}
return true;
```

C

```java
for (int x = 2; x < n; x++) {
    if (condition) {
        return true;
    }
}
return false;
```

D

```java
for (int x = 2; x < n; x++) {
    if (!condition) {
```
Loops and Arrays

return true;
}
else {
    return false;
}

Part Two

All of the implementations above are a tad inefficient, so let’s make them slightly more efficient! To do so, we will change the loop’s stopping condition. The stopping condition for all the implementations above is $x < n$. Which of the following stopping conditions are correct and would make the code more efficient?

Hint: You only need to check the factors until the square root of $n$

- $x <= n * n$
- $x * x <= n$
- $x * x * x <= n$