CIS 210
Winter 2012 Midterm

Write your name at the bottom of each page before you begin. 1 point for each page.

1. [5 points] What does method q1 print?

```java
public static void q1() {
    int x = 6;
    int y = 4;
    int z = x + y;
    if (x > y) {
        x = x - y;
    }
    if (x > y) {
        z = 42;
    }
    z = z + x;
    System.out.println(z);
}
```

12
It looks like most people got this one without much problem.

2. [5 points] What does method q2 print?

```java
public static void q2() {
    int x = 5;
    int y = 0;
    while (x > 0) {
        y = y + x;
        x = x / 2;
    }
    System.out.println(y);
}
```

8
A few people got 5 or 7, which looks like it could be from miscalculating the number of times the loop is executed.
3. [5 points] What does method q3 print?

```java
public static boolean isB(char ch) {
    return ch == ' ';  
}
```

```java
public static int nb(String s) {
    int n = 0;
    for (int i=0; i < s.length(); ++i) {
        char ch = s.charAt(i);
        if (isB(ch)) {
            n++;
        }
    }
    return n;
}
```

```java
public static void q3() {
    String phrase = "mystery wrapped in an enigma";
    System.out.println( nb(phrase) );
}
```

4. Several people struggled with this one. The return ch == ' ' returns the boolean (true or false) value of the expression ch == ' ', which is true if the character is a space (blank) and false otherwise. The loop in method nb counts the number of blanks in the input string s.
4. [5 points] What does method q4 print?

```java
public static void magnify(int[] ar) {
    for (int i=0; i < ar.length; ++i) {
        if (ar[i] < 0) {
            ar[i] = 0 - ar[i];
        }
    }
    return;
}

public static void q4() {
    int[] weights = new int[] { -5, 5, -4, 6, -7, 3 };
    magnify(weights);
    int magSum = 0;
    for (int i=0; i < weights.length; ++i) {
        magSum += Math.abs(weights[i]);
    }
    System.out.println( magSum );
}
```

I'm seeing several wrong answers on this, and I haven't quite figured out what the points of confusion were yet. The main thing I was looking for in this question was whether you understood that, when we pass an array as an argument to a method, we are passing a reference to the array, and therefore the called method can change elements of the array in a way that is visible to the calling method. Method `magnify` changes all the negative values in the array into positive values, so `q4` is essentially summing their absolute values.
5. [12 points] Fill in Java code to complete the method mostlyPositive, consistent with the Javadoc header comment.

```java
/**
 * Determine whether array ar contains more positive values than negative values.
 * @param ar An array of integers
 * @return true if the number of positive values in ar is strictly more than the number of negative values in ar; false if the number of negative values is at least equal to the number of positive values.
 * Examples: mostlyPositive( [ 0, 1, 2, -2 ] ) == true
 * mostlyPositive( [ -1, -2, 0, 0, 3, 4 ] ) == false
 * mostlyPositive( [ ] ) == false
 */
public static boolean mostlyPositive( int[] ar ) {
    int posCount = 0;
    int negCount = 0;
    for (int i=0; i < ar.length; ++i) {
        if (ar[i] < 0) {
            ++negCount;
        } else if (ar[i] > 0) {
            ++posCount;
        }
    }
    return posCount > negCount;
}

Variations are possible, although I think most good solutions will be pretty close to this one.
```
6. [13 points] Write the method strictlyAscending, consistent with the Javadoc header comment.

```java
/**
 * Determine whether the integers in ar are a strictly ascending sequence,
 * i.e., each element (after the first) is greater than the element before.
 * @param ar An array of zero or more integers
 * @return true if the values are strictly increasing, otherwise false.
 * Examples: strictlyAscending( [ 0, 1, 2, 3, 4, 5 ] ) == true
 * strictlyAscending( [ 0, 1, 1, 2, 3, 4 ] ) == false
 * strictlyAscending( [ ] ) == true
 */
public static boolean strictlyAscending(int[] ar) {
    if (ar.length == 0) {
        return true;
    }
    int prev = ar[0];
    for (int i=1; i < ar.length; ++i) {
        int cur = ar[i];
        if (cur <= prev) {
            return false;
        }
        prev = cur;
    }
    return true;
}
```

I expect most answers will not do it this way, but will rather have something like

```java
if (ar[i] >= ar[i+1]) { return false; }
```

in the loop (and omit the special case for length of zero). That's fine, with the loop stopping at `length - 2` instead of the usual `length - 1`.

You may wonder why I did it this way, rotating values through `cur` (current) and `prev` (previous). It's actually an approach that isn't needed here, but will be needed when we get to list structures, where we won't be able to access both `a[i]` and `a[i+1]` in this “random access” fashion, but can still look at each value in sequence.