1. Expressions, 10 points. For each expression in the left-hand column, indicate its value in the right-hand column. Be sure to list a constant of appropriate type (e.g., 7.0 rather than 7 for a double, Strings in quotes).

Expression	Value
`3 + 3 * 8 - 2`	_____
`109 % 100 / 2 + 3 * 3 / 2.0`	_____
`1 - 3 / 6 * 2.0 + 14 / 5`	_____
`1 + "x" + 11 / 10 + " is" + 10 / 2`	_____
`10 % 8 * 10 % 8 * 10 % 8`	_____

2. Parameter Mystery, 12 points. Consider the following program.

```
public class Mystery {
    public static void main(String[] args) {
        String she = "it";
        String it = "her";
        String her = "you";
        String you = "she";

        saying(you, it, you);
        saying(it, her, she);
        saying(she, "you", her);
        saying(it, "him", "fred");
    }

    public static void saying(String it, String her, String she) {
        System.out.println(she + " can't take " + it + " with " + her);
    }
}
```

List below the output produced by this program.

3. If/Else Simulation, 12 points. Consider the following method.

```
public static void ifElseMystery(int a, int b) {
    if (a < b) {
        a++;
    }
    if (a < b || a % 2 == 1) {
        a++;
    } else {
        b++;
    }
    if (a >= b && a % 2 == 0) {
        b = b - 5;
    }
    System.out.println(a + " " + b);
}
```

For each call below, indicate what output is produced.

Method Call	Output Produced
ifElseMystery(5, 15);	_____
ifElseMystery(12, 5);	_____
ifElseMystery(7, 8);	_____
ifElseMystery(2, 3);	_____
ifElseMystery(1, -2);	_____
ifElseMystery(1, 1);	_____

4. While Loop Simulation, 12 points. Consider the following method:

```
public static void mystery(int n) {
    int x = 1;
    int y = 1;
    while (n > 2) {
        x = x + y;
        y = x - y;
        n--;
    }
    System.out.println(x);
}
```

For each call below, indicate what output is produced.

Method Call	Output Produced
mystery(1);	_____
mystery(3);	_____
mystery(5);	_____
mystery(7);	_____

5. Assertions, 15 points. You will identify various assertions as being either always true, never true or sometimes true/sometimes false at various points in program execution. The comments in the method below indicate the points of interest.

```
public static int mystery(int n) {
    int x = 2;
    // Point A
    while (x < n) {
        // Point B
        if (n % x == 0) {
            n = n / x;
            x = 2;
            // Point C
        } else {
            x++;
            // Point D
        }
    }
    // Point E
    return n;
}
```

Fill in the table below with the words ALWAYS, NEVER or SOMETIMES.

	x > 2	x < n	n % x == 0
Point A			
Point B			
Point C			
Point D			
Point E			

6. Programming, 10 points. Write a static method called checkPrime that takes an integer n as a parameter and that checks how many factors n has, returning true if it is prime (exactly 2 factors) and returning false otherwise. Recall that a factor is a number between 1 and n that goes evenly into n and that prime numbers are those that have exactly two factors (1 is not a prime). So if we make the following call:

```
boolean test = checkPrime(24);
```

The method should produce the following two lines of output:

```
factors of 24 = 1, 2, 3, 4, 6, 8, 12, 24
Total factors = 8
```

and the variable test would be set to false because this number does not have exactly 2 factors. Below are three other sample calls:

checkPrime(1)	checkPrime(5)	checkPrime(25)
factors of 1 = 1	factors of 5 = 1, 5	factors of 25 = 1, 5, 25
Total factors = 1	Total factors = 2	Total factors = 3
returns false	returns true	returns false

Notice that for these examples the method returns true only for 5 because it is the only prime. You must exactly reproduce the format of these logs. You may assume that the value passed to your method is greater than 0.

7. File Processing, 10 points. Write a static method called printDuplicates that takes an input scanner as a parameter and that examines the tokens in the scanner, printing tokens that are duplicated sequentially. Your method should examine the tokens looking for consecutive occurrences of the same token, printing each duplicated token along with how many times it appears consecutively. Non-repeated tokens are not printed. For example, if a scanner called input contains the following tokens:

```
hello how how are you you you you I I I am Jack's Jack's smirking
smirking smirking smirking smirking revenge bow wow wow yippee yippee
yo yippee yippee yay yay yay one fish two fish red fish blue fish
```

and we make the following call:

```
printDuplicates(input);
```

then the method would produce the following output:

```
how*2
you*4
I*3
Jack's*2
smirking*5
wow*2
yippee*2
yippee*2
yay*3
```

You are to exactly reproduce the format of this output. You may assume that the scanner contains at least one token of input. Notice that line breaks in the input are not meaningful. You may not construct any extra data structures to solve this problem.

8. Arrays, 10 points. Write a static method called isPairwiseSorted that takes an array of integers as a parameter and that returns whether or not the array is pairwise sorted. An array is considered to be pairwise sorted if it contains a sequence of pairs where each pair is in sorted (nondecreasing) order. For example, if a variable list is defined as follows:

```
int[] list = {3, 8, 2, 15, -3, 5, 2, 3, 4, 4};
             |  |  |  |    |  |  |  |  |  |
             +--+  +--+    +--+  +--+  +--+
             pair  pair    pair  pair  pair
```

then the call isPairwiseSorted(list) would return true because the array is composed of a sequence of pairs that are each in sorted order ((3, 8) followed by (2, 15), followed by (-3, 5), and so on). If the array has an odd length, then your method should ignore the value at the end. Below are several examples of what value would be returned for a given array.

Array passed as parameter	Value Returned
{}	true
{6}	true
{4, 12}	true
{8, 5}	false
{6, 12, 4}	true
{3, 8, 2, 15, -3, 5, 2, 3, 4, 4, 3, 1}	false
{8, 13, 92, 92, 4, 4, 1}	true
{1, 3, 5, 7, 9, 8}	false

You may not construct any extra data structures to solve this problem.

9. Programming, 9 points. Write a static method called samePattern that returns true or false depending upon whether two strings have the same pattern of characters. More precisely, two strings have the same pattern if they are of the same length and if two characters in the first string are equal if and only if the characters in the corresponding positions in the second string are also equal. Below are some examples of patterns that are the same and patterns that differ (keep in mind that the method should return the same value no matter what order the two strings are passed).

1st String	2nd String	Same Pattern?
""	""	true
"a"	"x"	true
"a"	"ab"	false
"ab"	"ab"	true
"aa"	"xy"	false
"aba"	"+-+"	true
"---"	"aba"	false
"abcabc"	"zodzod"	true
"abcabd"	"zodzoe"	true
"abcabc"	"xxxxxx"	false
"aaasssccccn"	"aaabbbcccd"	true
"asasasasas"	"xyxyxyxyxy"	true
"ascneencsa"	"aeiouuoiea"	true
"aaasssccccn"	"aaabbbcccd"	true
"asasasasas"	"xxxxxyyyyy"	false
"ascneencsa"	"aeiouaeiou"	false
"aaasssccccn"	"xxxyyyzzzz"	false
"aaasssiiii"	"gggdddfffh"	false

Your method should take two parameters: the two strings to compare. You are allowed to create new strings, but otherwise you are not allowed to construct extra data structures to solve this problem (no array, ArrayList, Scanner, etc). You are limited to the string methods on the cheat sheet.