1. Expressions
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, true or false for a boolean).

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 + 4 * 5 / 2</td>
<td></td>
</tr>
<tr>
<td>13 % 5 + 43 % (11 % 3)</td>
<td></td>
</tr>
<tr>
<td>1.5 * 3.0 + 25.0 / 10.0</td>
<td></td>
</tr>
<tr>
<td>7 / 2 != 123 / 12 % 7</td>
<td></td>
</tr>
<tr>
<td>5 / 2 + 123 / 10 / 10.0</td>
<td></td>
</tr>
<tr>
<td>5 + 2 + &quot;(1 + 1)&quot; + 4 + 2 * 3</td>
<td></td>
</tr>
</tbody>
</table>

2. Parameter Mystery
At the bottom of the page, write the output produced by the following program, as it would appear on the console.

```java
class ParameterMystery {
    public static void main(String[] args) {
        String p = "cause";
        String q = "support";
        String r = "troops";
        String support = "hillary";
        String cause = "rudy";

        troops(p, q, r);
        troops(q, r, p);
        troops(support, p, cause);
        troops(r, "p", support);
        troops(q, "cause", q);
    }

    public static void troops(String r, String p, String q) {
        System.out.println(q + " gave " + r + " to the " + p);
    }
}
```
3. **If/Else Simulation**

For each call below to the following method, write the output that is produced, as it would appear on the console:

```java
public static void mystery(int n) {
    System.out.print(n + " ");
    if (n > 10) {
        n = n / 2;
    } else {
        n = n + 7;
    }
    if (n * 2 < 25) {
        n = n + 10;
    }
    System.out.println(n);
}
```

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery(40);</td>
<td></td>
</tr>
<tr>
<td>mystery(8);</td>
<td></td>
</tr>
<tr>
<td>mystery(0);</td>
<td></td>
</tr>
<tr>
<td>mystery(12);</td>
<td></td>
</tr>
<tr>
<td>mystery(20);</td>
<td></td>
</tr>
</tbody>
</table>

4. **While Loop Simulation**

For each call below to the following method, write the value that is returned:

```java
public static int mystery(int x) {
    int a = 1;
    int c = 0;
    while (x > 0) {
        a = x % 2;
        if (a == 1) {
            c++;
        }
        x = x / 2;
    }
    return c;
}
```

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery(2);</td>
<td></td>
</tr>
<tr>
<td>mystery(-1);</td>
<td></td>
</tr>
<tr>
<td>mystery(7);</td>
<td></td>
</tr>
<tr>
<td>mystery(18);</td>
<td></td>
</tr>
<tr>
<td>mystery(43);</td>
<td></td>
</tr>
</tbody>
</table>
5. Assertions

For the following method, identify each of the three assertions in the table below as being either ALWAYS true, NEVER true or SOMETIMES true / sometimes false at each labeled point in the code. (can abbreviate as A/N/S)

```java
public static int threeHeads() {  // Counts coin tosses till we get heads 3x in a row.
    Random rand = new Random();
    int flip = 1;
    int heads = 0;
    int count = 0;

    // Point A
    while (heads < 3) {
        // Point B
        flip = rand.nextInt(2);   // flip coin
        if (flip == 0) {  // heads
            heads++;
            // Point C
        } else {          // tails
            // Point D
            heads = 0;
        }
        count++;
    }
    // Point E
    return count;
}
```

<table>
<thead>
<tr>
<th>Point</th>
<th>flip == 0</th>
<th>heads == 0</th>
<th>flip &gt; heads</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. Programming

Write a static method named `printTwoDigit` that accepts a `Random` object and an integer `n` as parameters and that prints a series of `n` randomly generated two-digit numbers. The method should use the `Random` object to select numbers in the range of 10 to 99 inclusive where each number is equally likely to be chosen. After displaying each number that was produced, the method should indicate whether the number 42 was ever selected ("we saw a 42!") or not ("no 42 was seen."). You may assume that the value of `n` passed is at least 0.

The following table shows two sample calls and their output:

<table>
<thead>
<tr>
<th>Call</th>
<th>Random r = new Random(); printTwoDigit(r, 4);</th>
<th>Random r = new Random(); printTwoDigit(r, 7);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>next = 52</td>
<td>next = 83</td>
</tr>
<tr>
<td></td>
<td>next = 10</td>
<td>next = 29</td>
</tr>
<tr>
<td></td>
<td>next = 96</td>
<td>next = 42</td>
</tr>
<tr>
<td></td>
<td>next = 86</td>
<td>next = 22</td>
</tr>
<tr>
<td></td>
<td>no 42 was seen.</td>
<td>next = 91</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next = 36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next = 73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>we saw a 42!</td>
</tr>
</tbody>
</table>
7. Programming (15 points)

In this question, we'll address the following problem: Can a cash register containing a given amount of pennies (1-cent coins) and a given amount of nickels (5-cent coins) give a customer a given exact amount of cents of change? For example, if there are 3 pennies and 5 nickels in the cash register, is it possible to give exactly 19 cents of change? (No.) If there are 2 pennies and 7 nickels in the register, is it possible to give exactly 26 cents of change? (Yes.)

Write a static method named `canMakeChange` that accepts three integer parameters representing the number of pennies in the cash register, the number of nickels in the cash register, and the desired amount of change to make. The method should return `true` if the coins in the register can produce this exact amount of change, and `false` if not. The coins in the register must be able to exactly produce the desired amount of change in order to return `true`; for example, if the register contains 0 pennies and 100 nickels, it is not able to exactly produce 8 cents of change.

The following are several sample calls to your method and the values they should return. You may assume that no negative parameter values are passed, but otherwise your method should work with any values passed.

<table>
<thead>
<tr>
<th>Call</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>canMakeChange(3, 4, 12)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>canMakeChange(1, 5, 26)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>canMakeChange(24, 2, 31)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>canMakeChange(87, 19, 134)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>canMakeChange(0, 0, 0)</code></td>
<td>true</td>
</tr>
<tr>
<td><code>canMakeChange(1, 1, 9)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>canMakeChange(2, 7, 8)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>canMakeChange(4, 3, 39)</code></td>
<td>false</td>
</tr>
<tr>
<td><code>canMakeChange(3, 80, 14)</code></td>
<td>false</td>
</tr>
</tbody>
</table>

8. Programming

Write a static method named `consecutiveDigits` that accepts an integer `n` as a parameter and that returns the highest number of consecutive digits in a row from `n` that have the same value. For example, the number 3777785 has four consecutive occurrences of the number 7 in a row, so the call `consecutiveDigits(3777785)` should return 4.

For many numbers the answer will be 1 because they don't have any adjacent digits that match. Below are sample calls on the method. You are not allowed to use a `String` to solve this problem. You may assume that the value passed to the method is greater than or equal to 0.

<table>
<thead>
<tr>
<th>Call</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>consecutiveDigits(0)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>consecutiveDigits(18)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>consecutiveDigits(394)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>consecutiveDigits(99)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>consecutiveDigits(8229)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>consecutiveDigits(8823)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>consecutiveDigits(777)</code></td>
<td>3</td>
</tr>
<tr>
<td><code>consecutiveDigits(82888)</code></td>
<td>3</td>
</tr>
<tr>
<td><code>consecutiveDigits(711171)</code></td>
<td>4</td>
</tr>
<tr>
<td><code>consecutiveDigits(233333888)</code></td>
<td>5</td>
</tr>
</tbody>
</table>
1. **Expressions**

   **Expression** | **Value**
   --- | ---
   $3 + 4 \times 5 / 2$ | 13
   $13 \mod 5 + 43 \mod (11 \mod 3)$ | 4
   $1.5 \times 3.0 + 25.0 / 10.0$ | 7.0
   $7 / 2 != 123 / 12 \mod 7$ | false
   $5 / 2 + 123 / 10 / 10.0$ | 3.2
   $5 + 2 + "(1 + 1)" + 4 + 2 \times 3$ | "7(1 + 1)46"

2. **Parameter Mystery**

   troops gave cause to the support  
   cause gave support to the troops  
   rudy gave hillary to the cause  
   hillary gave troops to the p  
   support gave support to the cause

3. **If/Else Simulation**

   **Method Call** | **Output**
   --- | ---
   mystery(40); | 40 20
   mystery(8); | 8 15
   mystery(0); | 0 17
   mystery(12); | 12 16
   mystery(20); | 20 20

4. **While Loop Simulation**

   **Method Call** | **Value Returned**
   --- | ---
   mystery(2); | 1
   mystery(-1); | 0
   mystery(7); | 3
   mystery(18); | 2
   mystery(43); | 4

5. **Assertions**

   | flip == 0 | heads == 0 | flip > heads |
   --- | --- | ---
   Point A | NEVER | ALWAYS | ALWAYS |
   Point B | SOMETIMES | SOMETIMES | SOMETIMES |
   Point C | ALWAYS | NEVER | NEVER |
   Point D | NEVER | SOMETIMES | SOMETIMES |
   Point E | ALWAYS | NEVER | NEVER |

6. **Programming (one solution shown)**

   ```java
   public static void printTwoDigit(Random r, int n) {
       boolean seen42 = false;
       for (int i = 1; i <= n; i++) {
           int number = r.nextInt(90) + 10;
           System.out.println("next = " + number);
           if (number == 42) {
               seen42 = true;
           }
       }
       if (seen42) {
           System.out.println("we saw a 42!");
       } else {
           System.out.println("no 42 was seen.");
       }
   }
   ```
7. Programming (six solutions shown)

```java
public static boolean canMakeChange(int pennies, int nickels, int cents) {
    if (cents % 5 > pennies) {
        return false;
    } else if (pennies + 5 * nickels < cents) {
        return false;
    } else {
        return true;
    }
}

public static boolean canMakeChange(int pennies, int nickels, int cents) {
    while (cents >= 5 && nickels > 0) {
        cents -= 5;
        nickels--;
    }
    while (cents > 0 && pennies > 0) {
        cents--;
        pennies--;
    }
    return cents == 0;
}

public static boolean canMakeChange(int pennies, int nickels, int cents) {
    for (int p = 0; p <= pennies; p++) {
        for (int n = 0; n <= nickels; n++) {
            if (p + 5 * n == cents) {
                return true;
            }
        }
    }
    return false;
}

public static boolean canMakeChange(int pennies, int nickels, int cents) {
    if (nickels * 5 >= cents) {
        return (pennies >= cents % 5);  // enough nickels to cover all except % 5 part
    } else {
        return (pennies >= cents - nickels * 5);   // not enough nickels; need pennies
    }
}

public static boolean canMakeChange(int pennies, int nickels, int cents) {
    cents -= Math.min(nickels * 5, cents - cents % 5);
    return cents - pennies <= 0;
}

public static boolean canMakeChange(int pennies, int nickels, int cents) {
    return (pennies >= cents % 5) && (pennies + 5 * nickels >= cents);
}
```

8. Programming (one solution shown)

```java
public static int consecutiveDigits(int n) {
    int count = 1;
    int last = -1;    // dummy value; anything but 0-9
    int max = 1;
    while (n != 0) {
        if (n % 10 == last) {
            count++;
        } else {
            count = 1;
        }
        if (count > max) {     // or,  max = Math.max(max, count);
            max = count;
        }
        last = n % 10;
        n = n / 10;
    }
    return max;
}
```