To approach these problems, we can use $P(\text{MMDB)(AS})$

1. Parentheses $(\cdot)$: compute everything inside of the $(\cdot)$
2. Mod or multiplication or division: whichever comes first going left to right
3. Addition or subtraction: whichever comes first going from left to right

**Example:**

1. $(6-4) \times 3 + 12 / 5 \% 2$
   - $6-4 = 2$
   - $2 \times 3 = 6$
   - $12 / 5 = 2.4$
   - $2.4 \% 2 = 0$
   - $6 + 0 = 6$

2. $3 \times 0.5 + 1 + "7" + 7 + (6-2)$
   - $3 \times 0.5 = 1.5$
   - $1 + "7" = 1 + 7 = 8$
   - $1.5 + "7" = 1.5 + 7 = 8.5$
   - $8.5 + 7 + 4 = 20$
   - $20 + "7 + 4" = 20 + 7 + 4 = 21$
2.5 * 3 ≥ 10 % 6 \[ 11 \ 13/4 \] != 3

1. When approaching these problems with Boolean conditions, we want to solve until we have values that are comparable.

10 % 6 = 4 since 10 divided by 6 is 1 remainder 4

7.5 ≥ 10 % 6 \[ 11 \ 13/4 \] != 3

2. Now, we have comparable values. 7.5 ≥ 4, so this is true.

7.5 ≥ 4 \[ 11 \ 13/4 \] != 3

3. 13/4 = 3 since integer division rounds down.

true \[ 11 \ 13/4 \] != 3

4. Since 3 = 3, this is false.

true \[ 11 \ 3 \] != 3

5. when you or true and false, the outcome is true. When you and true and false, the outcome is false.

true \[ 11 \ false \] = true

true && false = false

true \[ 11 \ false \] = true

true && false = false
public static int m(int x, int y) {
    while (x > 0 && y > 0) {
        x = x - y;
        y--;  
        System.out.print(x + " ", "");
        System.out.print(y + " ");
    }
    return y;
}

m(14, 9):
while (14 > 0 && 9 > 0) {
    x = 14 - 9;
    y--;  
    x = 5;
    y = 8;
}

m(5, 0):
while (5 > 0 && 0 > 0) {
    x = 5 - 8;
    y--;  
    x = -3;
    y = 7;
}

m(-17, -8):
return 7

m(11, -3):
return -3

Writing out what happens for each iteration of the loop can help us keep track of our x and y values.

We can ignore the print statement since we only care about the return value we do not enter the while loop since y > 0 and x < 0. Therefore, we return 0.

We do not enter the while loop since y > 0 and x > 0. Therefore, we return -8.

We do not enter the while loop since y > 0. Since we return -3, this would be odd.
m(10, 10)
while (10 > 0 && 10 > 0) {
    x = 10 - 10;  \to x = 0
    y --; \to y = 9
    return 9
}

public static void mystery(int x, int y) {
    int z = 0;
    // Point A
    while (x < y) {
        // Point B
        z + +;
        if (z % 2 == 0) {
            x = x * 2;
            // Point C
        } else {
            y --;
            // Point D
        }
    }
    // Point E
    System.out.println(z);
}

A: We have not entered the while loop yet, so we do not know if these conditions are true
B: Since we entered the while loop, we will only reach point B if \( x \leq y \)
C: Since we entered the if statement, \( z \% 2 == 0 \)
at this point
D: Since we skipped past the if statement and entered the else branch, then \( z \% 2 \neq 0 \)
E: We exited the loop at point E, so \( x \leq y \)

- At Point A, \( x \leq y \) must be true
- At Point B, \( x \leq y \) must be true
- At Point C, \( z \% 2 == 0 \) must be true
- At Point D, \( z \% 2 == 0 \) must be true
- At Point E, \( x \leq y \) must be true
2. Array Code Tracing

Consider the following method:

```java
public static int[] mystery(int[][] list) {
    int[] result = new int[list.length];
    for (int i = 0; i < list.length; i++) {
        int n = 1;
        for (int j = 0; j < list[0].length; j++) {
            n *= list[i][j];
            result[i] = n;
        }
        return result;
    }
}
```

For each element in an “inner” array, multiply them together. Creating a new array whose length is the number of rows in the 2D array “list” array traversal going through each row in the outer loop, and each column in the inner loop.

Part A: Consider the following code:

```java
int[][] arr = {{3, 15, 1},
               {-8, 1, 7},
               {7, 11, 0},
               {-1, -9, 4}};
int[] result = mystery(arr);
```

What are the contents in `arr` after this code is executed?

```
{{3, 15, 1},
 {-8, 1, 7},
 {7, 11, 0},
 {-1, -9, 4}}
```

The contents in `arr` are unchanged. The method only uses the values in `arr` without modifying the 2D array in any way.

Part B: Consider the following code:

```java
int[][] arr = {{3, 15, 1}, 3 * 15 * 1 = 45
               {-8, 1, 7}, 8 * 1 * 7 = -56
               {7, 11, 0}, 7 * 11 * 0 = 0
               {-1, -9, 4}}; 1 * -9 * 4 = 36
int[] result = mystery(arr);
```

What are the contents in `result` after this code is executed?

```
{45, -56, 0, 36}
```

Part C: Which of the following best describes what the method mystery does? (Choose one)

- Returns a new array holding the last element in each row of `arr`, and modifies `arr` to contain the products of each row.
- Returns a new array holding the last element in each column of `arr`, and modifies `arr` to contain the products of each column.
- Returns a new array holding the products of the elements in each row of `arr`, and fills `arr` with 1's.
- Returns a new array holding the products of the elements in each column of `arr`, and fills `arr` with 1's.
- Returns a new array containing the products of the elements in each row of `arr`, leaving `arr` unchanged.
- Returns a new array containing the products of the elements in each column of `arr`, leaving `arr` unchanged.
Option 1:
Incorrect, because the purpose of the "mystery" method is not to get the last element in each row of arr. Also, arr is not modified.

Option 2:
Incorrect, for nearly the same reason that "mystery" is not getting the last element in each column. Again, arr is not modified.

Option 3:
The first sentence is correct. The new array does contain the products in each row. But, arr does not become replaced with all 1's.

Option 4:
Correct, result contains the products of each row in arr.

Option 5:
Incorrect, result does not contain the product of each column, because the array traversal loops through the 2D array in the following way:

1) Start at $i = 0$
2) Start at $n = 1$
3) Start at $j = 0$
4) Loop through arr $[0].length$, which is the length of 1 row in arr.
5) For each element (arr $[i][j]$), multiply them together.
6) After the $j$ loop finishes going through indices $[0][0], [0][1], [0][2]$, set the index at result $[i] = n$
7) Repeat 2-6 for each value of $i$, when $i < list.length$, which is the number of rows in arr.
3. Debugging

Consider a static method called `battle` that simulates a battle between two players, which takes two parameters:

- `int minDamage` - the minimum amount of damage a player can inflict upon the other (guaranteed to be at least 0)
- `int maxDamage` - the maximum amount of damage a player can inflict upon the other (guaranteed to be greater than `minDamage`)

The "health" of a player is represented by a number initially set to 100. Each player randomly attacks the other, subtracting damage from the attacked player's health, until one of the player's health falls below 1. The next player to attack is randomly determined, and the damage inflicted is a random number between the minimum damage value and the maximum damage value (inclusive).

For example, suppose the following call was made:

```
battle(20, 50);
```

This call to a correct implementation of the method might produce output like the following. (Due to the randomness involved in the method, this exact output may not be produced every time it is run.):

```
Let's get ready to rumble!!!
Player 2 attacks! 35 damage...P1: 65, P2: 100
Player 1 attacks! 37 damage...P1: 65, P2: 63
Player 1 attacks! 43 damage...P1: 65, P2: 20
Player 2 attacks! 21 damage...P1: 44, P2: 20
Player 2 attacks! 43 damage...P1: 1, P2: 20
Player 2 attacks! 32 damage...P1: -31, P2: 20
Player 2 wins!
```

Consider the following proposed buggy implementation of `battle`:

```
public static void battle(int minDamage, int maxDamage) {
    System.out.println("Let's get ready to rumble!!!");
    Random r = new Random(); // consider: true && false => true
    int player = 0;
    int playerOneHealth = 100;
    int playerTwoHealth = 100;
    while (playerOneHealth > 0 && playerTwoHealth > 0) {
        int damage = r.nextInt(maxDamage - minDamage + 1); // remember: r.nextInt(max-min+1) + min
        player = r.nextInt(2) + 1; // consider: true && false => false
        if (player == 1) {
            playerTwoHealth -= damage;
        } else {
            playerOneHealth -= damage;
        }
    }
    System.out.println("Player " + player + " attacks! " + damage + " damage");
    System.out.println("P1: " + playerOneHealth + ", P2: " + playerTwoHealth);
    System.out.println("Player " + player + " wins!");
}
```
This implementation contains **two bugs** that are causing it to not work as intended!

For the same input as before, the buggy implementation might produce the following output:

```java
1 public static void battle(int minDamage, int maxDamage) {
2     System.out.println("Let's get ready to rumble!!!");
3     Random r = new Random();
4     int player = 0;
5     int playerOneHealth = 100;
6     int playerTwoHealth = 100;
7     while (playerOneHealth > 0 && playerTwoHealth > 0) {
8         int damage = r.nextInt(maxDamage - minDamage + 1) + minDamage;
9         player = r.nextInt(2) + 1;
10        if (player == 1) {
11             playerTwoHealth -= damage;
12        } else {
13            playerOneHealth -= damage;
14        }
15        System.out.print("Player " + player + " attacks! " + damage + " damage");
16        System.out.println("P1: " + playerOneHealth + ", P2: " + playerTwoHealth);
17    }
18    System.out.println("Player " + player + " wins!");
19 }
```

**Your task:** Annotate (write on) the code below to indicate how you would fix the two bugs. You may add (using arrows to indicate where to insert), remove (by crossing out), or modify (with a combination) any code you choose. However, the fix should not require a lot of work.

You must **correctly identify** both of the lines with issues, or **correctly identify and fix one** of the bugs for an S grade.

You must **correctly identify** both of the lines with the bugs and **correctly fix** both of the bugs for an E grade.
```java
public static int longWords(scanner input, int numWords) {

  // If we look at the first call to longWords, the output has 5 "Next word?" lines but only 4 "— more words..." output
  // This indicates that we might need to fencepost → we will have one initial scanner call outside of the loop
  System.out.print("Next word? ");
  String word = input.next();

  // Because we need to keep track of the longest word throughout iterations, we can create variables to store these values
  String longest = word;
  int totalChars = word.length();

  // fenceposting
  for (int i = 1; i < numWords; i++) {
    System.out.println((numWords - i) + " more words...");
    System.out.print("Next word? ");
    word = input.next();
    // we can add the length of each word to the character count variable to keep a running total
    totalChars += word.length();
    // we can also update the longest word if necessary per iteration
    if (word.length() > longest.length())
      longest = word;
  }

  System.out.println("Longest word: "+longest);
  return totalChars;
}
5. General Programming 2

Write a static method named `gumballTricks` that accepts a `Random` object and an integer `n` that represents the number of tricks as parameters. Your method should use the `Random` object to randomly choose a trick from Gumball's repertoire: spin, bang!, and boop. Each outcome should be equally likely. Your method should print out each of the randomly-generated tricks followed by a space, and then both print and return the greatest number of spins that occurred in a row.

This method:
1) Takes in 2 parameters: `Random`, `int`
2) Prints values out
3) Returns an `int`: max # of spins

Assuming that the following variable has been initialized:

```java
Random r = new Random();
```

Here are some example calls to the method with their resulting console output and return value:

<table>
<thead>
<tr>
<th>Call</th>
<th>Console Output</th>
<th>Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gumballTricks(r, 8);</code></td>
<td>spin boop bang! spin spin spin bang! spin Run of 3 spins after 8 tricks.</td>
<td>3</td>
</tr>
<tr>
<td><code>gumballTricks(r, 5);</code></td>
<td>bang! boop boop bang! boop Run of 0 spins after 5 tricks.</td>
<td>0</td>
</tr>
<tr>
<td><code>gumballTricks(r, 2);</code></td>
<td>spin bang! Run of 1 spins after 2 tricks.</td>
<td>1</td>
</tr>
<tr>
<td><code>gumballTricks(r, 10);</code></td>
<td>bang! spin spin boop spin spin spin spin bang! boop Run of 4 spins after 10 tricks.</td>
<td>4</td>
</tr>
<tr>
<td><code>gumballTricks(r, 3);</code></td>
<td>bang! bang! boop Run of 0 spins after 3 tricks.</td>
<td>0</td>
</tr>
<tr>
<td><code>gumballTricks(r, 1);</code></td>
<td>bang! Run of 0 spins after 1 tricks.</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that if there is only one trick, your final line should end with "1 tricks" (not 1 trick). Do not check if there is only 1 trick

You must exactly reproduce the format of the console output shown above, though the actual output may differ due to randomness. It is okay for the line of tricks to end with a space. You may assume that the integer passed as a parameter to your method is greater than 0.

You may not construct any extra data structures (e.g. arrays, ArrayLists) to solve this problem.

We only need to use some counter variables to solve this problem.
In my method header, I know I'll take in a Random, and an int, as seen in the description and the example method calls. Additionally, I can see that I should return an int, which represents the max number of spins.

```java
public static int gumballTrick(Random r, int n)
```

Given my int parameter, this represents the total number of tricks that gumball will perform. Therefore, I want to construct a `for`-loop that will repeat for the given number of times.

```java
for(int i=0; i<n; i++)
```

Now thinking about how I will keep track of the number of "spins" in a row, I will declare a variable before the `for` loop to count this value for each iteration of the loop.

```java
int spinsInRow=0
```

However, I will need to create an additional variable that will keep track of the greatest number of spins in a row.

```java
int maxSpinsInRow=0
```

Moving into the `for`-loop, next I need to consider how to make each trick equally likely. This can be represented by a random number generation with a range of 0 (inclusive) to 3 (exclusive) since each trick can be represented by a different value in the range.

```java
int trick = r.nextInt(3)
```

Next, I will need to construct a conditional structure to check whether the trick was "spin" or not. I can arbitrarily choose "0" to represent "spin," "1" for "bang!" and "2" for "boop," but these must stay consistent for all iterations of the loop.

If the value returned by my `nextInt(3)` call is 0, I will print out "spin." Also, I will update my "spin" counter by 1, and also check if it is greater than the max value so far.

```java
1  2
```

Else, I will reset the counter back to 0, and print out either "bang!" or "boop." Lastly, after the `for`-loop, I'll print out the max # of spins, and then return it.
6. Array Programming

Write a static method named \texttt{weave} that accepts two arrays of integers as a parameter and that returns a new array that is the result of alternating the values from the two arrays, starting with the first value of the first array. For example, if variables named \texttt{a1} and \texttt{a2} store the following values:

\begin{verbatim}
int[] a1 = {1, 2, 3};
int[] a2 = {4, 5, 6};
\end{verbatim}

then the call of \texttt{weave(a1, a2)} would return a new array containing the following values:

\begin{verbatim}
new array: [1, 4, 2, 5, 3, 6]
\end{verbatim}

It is possible that the two arrays may have different lengths, in which case after running out of values from the shorter array, the remaining slots of the result array are filled with the leftover elements of the longer array. For example, if variables named \texttt{a1} and \texttt{a2} store the following values:

\begin{verbatim}
int[] a1 = {1, 2, 3, 4, 5, 6};
int[] a2 = {7, 8, 9};
\end{verbatim}

then the call of \texttt{weave(a1, a2)} would return a new array containing the following values:

\begin{verbatim}
[1, 7, 2, 8, 3, 9, 4, 5, 6]
\end{verbatim}

You are not permitted to create any additional data structures (e.g. arrays, ArrayLists, Strings) other than the result array that you return. Only create one new array!
My Annotated Solution:

0) write method header: parameters - int[] al, int[] a2; return: int[

public static int[] weave (int[] al, int[] a2) {

    // since we are returning an int[], create a new int[]
    // this int[] will contain all elements from al and a2, so its length should be al.length + a2.length
    int[] result = new int[al.length + a2.length];

    ① weave: we should first determine which array (al or a2) has the shortest length;
    // we will weave until whichever array first runs out of elements

    al: [1, 2, 3] a2: [4, 5, 6] notice how al[0] is mapped to result[0]
    a2[0] is mapped to result[0] => index i of al is mapped to
    al [1] is mapped to result [2] a2[1] is mapped to result [3] index i of a2 is mapped to
    a2[2i] of result (if i = 1, result index = 1, 2 = 2)
    a2[i+1] of result (if i = 1, result index = 1, 2 + 1 = 3)

    for (int i = 0; i < Math.min (al.length, a2.length); i++) {
        result[2i] = al[i];
        result[2i+1] = a2[i];
    }

    ③ attach excess elements of one array after the weave in result

    al: [1, 2, 3, 4, 5, 6] a2: [7, 8, 9]
    0 1 2 3 4 5 0 1 2
    0 1 2 3 4 5 0 1 2
    result: [1, 7, 2, 8, 3, 9, 4, 5, 6] ④ we need to determine whichever array is longer so that we can
    0 1 2 3 4 5 0 1 2 attach those elements to the end
    of weave in result

    L notice how our starting index to attach is two times the minimum array length
    (3 * 2 = 6), to iterate through the result, we would start from this index until
    result.length exclusive.

    L notice how the difference between index i of result and the corresponding
    index of the array with excess elements is the minimum array length; we can use
    "i - Math.min (al.length, a2.length)" to retrieve the element in the longer array

    for (int i = 2; Math.min (al.length, a2.length); i < result.length; i++) {
        if (al.length > a2.length) {
            result[i] = al[i - Math.min (al.length, a2.length)];
        } else {
            result[i] = a2[i - Math.min (al.length, a2.length)];
        }
    }

    return result; // return our int[] result
}
public static int[] weave(int[] a1, int[] a2) {
    int[] a3 = new int[a1.length + a2.length];
    int shorterLength = Math.min(a1.length, a2.length);
    for (int i = 0; i < shorterLength * 2; i++) {
        if (i % 2 == 0) {
            a3[i] = a1[i / 2];
        } else {
            a3[i] = a2[i / 2];
        }
    }
    if (a1.length > a2.length) {
        for (int i = shorterLength; i < a1.length; i++) {
            a3[i + shorterLength] = a1[i];
        }
    } else {
        for (int i = shorterLength; i < a2.length; i++) {
            a3[i + shorterLength] = a2[i];
        }
    }
    return a3;
}