Exercise 1: Bad Loop! (Code Tracing and Arrays)

We’ve seen that code tracing is an important technique to figure out what code does. We’ll see in this exercise that we also can use this technique to figure out why code isn’t working as desired. When tracing code, we are the computer! We do a line-by-line execution of the code accompanied by drawing and changing memory diagrams as well as writing the output that would be displayed. This helps us see what is happening to the information being used by and generated in a program.

The code shown below was supposed to set the `characters` array to hold the characters

```
 a * b * c * d * e * f * g * h * i * j *
```

However, the code does not work correctly.

```
char [] characters = new char [20];
for (int n = 0; n<characters.length; n++) {
    characters[n] = '*';
}

char a = 'a';
for (int n = 0; n < characters.length; n++) {
    while (a != 'k') {
        if (n%2==0) characters[n] = a;
        a = (char)(a + 1);
    }
}
```

First, trace the code to find out what the final values in the `characters` array actually are.

Then, fix the code so that it works as intended.
Exercise 2: Star Pictures (Tracing and Coding Nested Loops)

For this exercise we’ll draw pictures one character at a time using nested loops. Note that the code we write is constrained by how a cursor moves in a console window: it moves from left to right within a line, and lines are drawn from top to bottom of the console window.

Part (a):

What is printed by the code shown below?

```
int size = 7;
for (int row=0; row<size; row++) {
    for (int col=0; col<row+1; col++) {
        System.out.print('*');
    }
    System.out.println();
}
System.out.println();
```

What if we set size to 3? To 8?

Part (b):

Using a laptop, write a program that draws one (or more) of the following pictures, after the user enters the value for size (the number of rows to draw). Be sure that your code works for any non-negative value of size. (If you choose one of the last three pictures, first decide how to handle even values of size).

```
*        *  *  *  *
***      ***  **  ***  **
*****    *****  ****  ****  ****
*******   ****  ****  *  ********
*********  ***  ***  ***  ********
********** **  **  ****  ****
***********  *  *  **  **  *****
```

---

-2-
Part (c):

Now write a new program that is similar to the one you wrote for Part b, but draws a hollow shape like these:

```
*   **********   *   **********   *
  * *   * *     * *   * *     * *
  *   *       *   *       *
  * *       * *       *
  *       *       *
**************   *   **************   *
```

Part (d):

Now design your own picture (some pattern of stars) and ask your neighbor to code it using nested loops.

```
*
*
*
*
*
*
*
*
```

Part (e):

If you have time, see if you can figure out how to draw shapes like this one:

```
**********
**  **
* * *
*   *
* * *
*   *
**  **
**********
```
Exercise 3: Lights-Out! (Logic & Programming)

For this exercise you will program a game for one player. The game is played on a row of lighted buttons, and the aim is to turn all the lights off. The tricky part is that pushing a button changes the on/off state of that button plus its two neighbors. Also, the first and last buttons are considered to be neighbors.

First, play the game on the laptops to see how it’s supposed to work.

Then look at the incomplete code on the next page.

Part (a)

Work in groups to write the missing code on paper (you won’t have laptops when you take the exam next week!)

If you finish early, see if you can come up with a strategy to win the game for a row of 4 buttons. How about a row of 5 buttons? A row of N buttons?

When everyone is ready, trade code with another group and see if you can find any errors.

Part (b)

Now each group use one of the laptops to complete and test your program.
```java
class LightsOut {

    public static void main(String[] args) {
        boolean[] game; // true means the light is on, false off
        int numLights;
        Scanner in = new Scanner(System.in);

        numLights = getNumLights(in);
        game = new boolean[numLights];

        // initialize so all lights are on
        for (int k=0; k<numLights; k++) {
            game[k] = true;
        }

        printGame(game);
        while (lightOn(game)) {
            int k = getButtonNum(in, numLights);
            pressButton(game, k-1);
            printGame(game);
        }
        System.out.println("Congratulations, you win!");
    }

    private static boolean lightOn(boolean[] game) {
        // return true iff at least 1 light is on
    }

    private static void pressButton(boolean[] game, int k) {
        // change the state of button[k] and its neighbors
    }

    private static void printGame(boolean[] game) {
        // print the numbers from 1 to number of lights
        // then print the lights (O for on, X for off)
    }

    private static int getNumLights(Scanner in) {
        // get input from the player
    }

    private static int getButtonNum(Scanner in, int max) {
        // get input from the player
    }
}
```