Solution 1: Simple JavaScript expressions, statements, and functions (10 points)

(1a)  
\[
\begin{align*}
5 \% 4 &- 4 \% 5 & 1 \\
7 < 9 &- 5 \&\& 3 \% 0 \equiv 3 & false \\
"B" + 3 \times 4 & \equiv "B12"
\end{align*}
\]

(1b)  "cabbage"

(1c)  "To care is human!"

Solution 2: Using graphics and animation (15 points)

```javascript
/* Constants */
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 300;
const CROSSBAR_LENGTH = 60;
const CROSSBAR_BREADTH = 20;
const TIME_STEP = 20;
const CROSS_SPEED = 2;

/* Main program */
function RedCross() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    let cross = createRedCross(CROSSBAR_LENGTH, CROSSBAR_BREADTH);
    gw.add(cross, gw.getWidth()/2, gw.getHeight()/2);
    let direction = randomReal(0, 360);
    let step = function() {
        cross.movePolar(CROSS_SPEED, direction);
    };
    setInterval(step, TIME_STEP); // return value can be ignored
    let clickAction = function(e) {
        if (gw.getElementAt(e.getX(), e.getY()) === null) return;
        direction = randomReal(0, 360);
    };
    gw.addEventListener("click", clickAction);
}
```
/**
 * Function: createRedCross
 * ----------------------
 * Constructs and returns a GCompound consisting of two
 * red rectangles—the first wide by narrow pixels in size, the
 * second narrow by wide pixels in size—such that their centers
 * overlap.
 */
function createRedCross(wide, narrow) {
    let cross = GCompound();
    cross.add(createFilledRectangle(wide, narrow, "Red"));
    cross.add(createFilledRectangle(narrow, wide, "Red"));
    return cross;
}

/**
 * Function: createFilledRectangle
 * ------------------------------
 * Constructs and returns a filled rectangle of the specified
 * width, height, and color (both border and fill). Note that
 * the rectangle is positioned so that it’s drawn relative to the
 * reference point of the GCompound, which is the GCompound’s center.
 */
function createFilledRectangle(width, height, color) {
    let rect = GRect(-width/2, -height/2, width, height);
    rect.setFilled(true);
    rect.setColor(color);
    return rect;
}

Solution 3: Strings (15 points)

/**
 * Function: spoonerism
 * --------------
 * Defines the spoonerism function according to the specifications
 * laid out in the first practice midterm.
 */
function spoonerism(phrase) {
    let sp1 = phrase.indexOf(' ');
    let sp2 = phrase.lastIndexOf(' ');
    let orig1 = phrase.substring(0, sp1);
    let orig2 = phrase.substring(sp2 + 1); // the +1 skips the " "!
    let middle = phrase.substring(sp1, sp2 + 1);

    let vp1 = findFirstVowel(orig1);
    let vp2 = findFirstVowel(orig2);
    let transformed1 = orig2.substring(0, vp2) + orig1.substring(vp1);
    let transformed2 = orig1.substring(0, vp1) + orig2.substring(vp2);
    return transformed1 + middle + transformed2;
}
/**
 * Function: findFirstVowel
 * ------------------------
 * Returns the index of the first lowercase vowel, or -1 if no lowercase vowel could be found.
 */
function findFirstVowel(str) {
    for (let i = 0; i < str.length; i++) {
        if (isEnglishVowel(str.charAt(i))) {
            return i;
        }
    }
}

/**
 * Function: isEnglishVowel
 * ------------------------
 * Returns true if and only if the provided string is of length 1, and its one character is a lowercase vowel.
 */
function isEnglishVowel(ch) {
    return ch.length == 1 && "aeiou".indexOf(ch) >= 0;
}

Solution 4: Arrays (15 points)

/**
 * Function: leaders
 * ---------------
 * Accepts an array of integers and returns a new array containing that array's leaders. The leaders are returned in the order they appear in the original array.
 */
function leaders(array) {
    let result = [];
    for (let i = 0; i < array.length; i++) {
        let include = true;  // note: include is part of the loop's test!
        for (let j = i + 1; include && j < array.length; j++) {
            include = array[i] > array[j];
        }
        if (include) {
            result.push(array[i]);
        }
    }
    return result;
}
Solution 5: Working with data structures (15 points)

/**
 * Predicate function: playerSmellsWumpus
 * --------------------------------------
 * Searches the cave just enough to decide whether
 * the player is within one or two rooms of the wumpus.
 * We assume the player and wumpus are guaranteed to be
 * in distinct rooms.
 */

function playerSmellsWumpus(cave) {
    let room = cave.playerLocation;
    for (let i = 0; i < 3; i++) {
        let roomOneAway = cave.connections[room][i];
        if (roomOneAway === cave.wumpusLocation) return true;
        for (let j = 0; j < 3; j++) {
            let roomTwoAway = cave.connections[roomOneAway][j];
            if (roomTwoAway === cave.wumpusLocation) return true;
        }
    }
    return false;
}