Eric Roberts and Jerry Cain
CS 106J

Handout #51A
June 7, 2017

Answers to Practice Final #1

Review session: **Sunday, June 11, 6:00–8:00 P.M. (Gates B-12)**

Scheduled final: **Wednesday, June 14, 8:30–11:30 A.M. (Lathrop 282)**

Problem 1—Short answer (10 points)

1a) As written, the program leaves the array in the following state:

list

50	10	10	10	10

If you had wanted **mystery** to "rotate" the array elements, you would need to run the loop in the opposite order to ensure that no elements are overwritten, like this:

```
function mystery(array) {
   var tmp = array[array.length - 1];
   for (var i = array.length - 1; i > 0; i--) {
      array[i] = array[i - 1];
   }
   array[0] = tmp;
}
```

1b) Calling **covfefe()** displays the value 28 on the console. The key to understanding this problem lies in figuring out which **x** and **y** values are used at each point. In the function returned by **puzzle**, the value of **x** comes from the closure and is therefore the value 17 passed to puzzle, and the value of **y** is the argument to the function **f**, which is 6. The body of the function computes 2 times **x** minus **y**, which is 28.

Problem 2—Simple graphics (15 points)

```
/*
 * Creates a GCompound object that represents a pie chart composed
 * of the data in the array.  The reference point of the GCompound
 * is the center of the circle.
 */

function createPieChart(r, data) {
   var gc = GCompound();
   var total = sumArray(data);
   var start = 0;
   for (var i = 0; i < data.length; i++) {
      var sweep = 360.0 * data[i] / total;
      var arc = GArc(-r, -r, 2 * r, 2 * r, start, sweep);
      arc.setFilled(true);
      arc.setFillColor(WEDGE_COLORS[i % WEDGE_COLORS.length]);
      gc.add(arc);
      start += sweep;
   }
   return gc;
}

/*
 * Returns the sum of the array.
 */

function sumArray(array) {
   var total = 0;
   for (var i = 0; i < array.length; i++) {
      total += array[i];
   }
   return total;
}
```

Problem 3—Interactive graphics (20 points)

```
/*
 * File: FifteenPuzzle.java
 * ------------------------
 * This program animates the Fifteen Puzzle.
 */

import "graphics";

/* Constants */

const SQUARE_SIZE = 60;
const GWINDOW_WIDTH = 4 * SQUARE_SIZE;
const GWINDOW_HEIGHT = 4 * SQUARE_SIZE;
const PUZZLE_FONT = "SansSerif-18";

/* This program simulates the classic Fifteen Puzzle */

function FifteenPuzzle() {
   var gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   initFifteenPuzzle(gw);
   var clickAction = function(e) {
      var obj = gw.getElementAt(e.getX(), e.getY());
      if (obj !== null) {
         if (tryToMove(gw, obj, SQUARE_SIZE, 0)) return;
         if (tryToMove(gw, obj, -SQUARE_SIZE, 0)) return;
         if (tryToMove(gw, obj, 0, SQUARE_SIZE)) return;
         if (tryToMove(gw, obj, 0, -SQUARE_SIZE)) return;
      }
   };
   gw.addEventListener("click", clickAction);
}

/*
 * Adds the numbered squares to the graphics window to create the
 * initial arrangement of the Fifteen Puzzle.
 */

function initFifteenPuzzle(gw) {
   var x = 0;
   var y = 0;
   for (var i = 1; i <= 15; i++) {
      gw.add(createNumberedSquare(i, SQUARE_SIZE), x, y);
      if (i % 4 === 0) {
         x = 0;
         y += SQUARE_SIZE;
      } else {
         x += SQUARE_SIZE;
      }
   }
}
```

☞

```
/*
 * Tries to move the object by the specified distance in the x
 * and y directions.  If it succeeds, the method returns true.
 * The +1 offset in the definitions of dx and dy is there to ensure
 * that getElementAt does not return this object.  We would not
 * require you to include that offset in your solution.
 */

function tryToMove(gw, obj, dx, dy) {
   var tx = obj.getX() + dx + 1;
   var ty = obj.getY() + dy + 1;
   if (tx < 0 || tx > gw.getWidth()) return false;
   if (ty < 0 || ty > gw.getHeight()) return false;
   if (gw.getElementAt(tx, ty) !== null) return false;
   obj.move(dx, dy);
   return true;
}

/*
 * Creates a numbered square consisting of a GCompound containing a
 * centered label.  The reference point is in the upper left corner.
 */

function createNumberedSquare(number, size) {
   var square = GCompound();
   var frame = GRect(0, 0, size, size);
   frame.setFilled(true);
   frame.setFillColor("LightGray");
   square.add(frame);
   var label = GLabel("" + number);
   label.setFont(PUZZLE_FONT);
   var x = (size - label.getWidth()) / 2;
   var y = (size + label.getAscent()) / 2;
   square.add(label, x, y);
   return square;
}
```

Problem 4—Strings (15 points)

```
/*
 * File: IsAnagram.js
 * ------------------
 * This file defines the isAnagram function from the practice final.
 */

function isAnagram(s1, s2) {
   var table1 = createFrequencyTable(s1);
   var table2 = createFrequencyTable(s2);
   for (var i = 0; i < table1.length; i++) {
      if (table1[i] !== table2[i]) return false;
   }
   return true;
}

/*
 * Creates a letter frequency table from the specified string.
 */

function createFrequencyTable(str) {
   var letterCounts = createArray(26, 0);
   for (var i = 0; i < str.length; i++) {
      var ch = str.charAt(i).toUpperCase();
      if (isLetter(ch)) {
         letterCounts[ch.charCodeAt(0) - "A".charCodeAt(0)]++;
      }
   }
   return letterCounts;
}

/*
 * Returns true if the character ch is a letter.
 */

function isLetter(ch) {
   return ch.length === 1 && ((ch >= "A" && ch <= "Z") ||
                              (ch >= "a" && ch <= "z"));
}

/*
 * Creates an array of n elements, each of which is initialized to value.
 */

function createArray(n, value) {
   var array = [ ];
   for (var i = 0; i < n; i++) {
      array.push(value);
   }
   return array;
}
```

Problem 5—Arrays (10 points)

```
/*
 * Returns an image that is twice the size of the original in each
 * dimension.  Each pixel in the original is replicated so that
 * it appears as a square of four pixels in the new image.
 */

function doubleImage(image) {
   var oldPixels = image.getPixelArray();
   var height = oldPixels.length;
   var width = oldPixels[0].length;
   var newPixels = createArray(2 * height, null);
   for (var i = 0; i < height; i++) {
      newPixels[2 * i] = createArray(2 * width, 0);
      newPixels[2 * i + 1] = createArray(2 * width, 0);
      for (var j = 0; j < width; j++) {
         var pixel = oldPixels[i][j];
         newPixels[2 * i][2 * j] = pixel;
         newPixels[2 * i][2 * j + 1] = pixel;
         newPixels[2 * i + 1][2 * j] = pixel;
         newPixels[2 * i + 1][2 * j + 1] = pixel;
      }
   }
   return GImage(newPixels);
}

/*
 * Creates an array of n elements, each of which is initialized to value.
 */

function createArray(n, value) {
   var array = [ ];
   for (var i = 0; i < n; i++) {
      array.push(value);
   }
   return array;
}
```

Problem 6—Working with data structures (15 points)

```
/*
 * Returns true if the player is one or two rooms away from the wumpus.
 */

function doesPlayerSmellAWumpus(cave) {
   var room = cave.playerLocation;
   for (var i = 0; i < 3; i++) {
      var oneRoomAway = cave.connections[room][i];
      if (oneRoomAway === cave.wumpusLocation) return true;
      for (var j = 0; j < 3; j++) {
         var twoRoomsAway = cave.connections[oneRoomAway][j];
         if (twoRoomsAway === cave.wumpusLocation) return true;
      }
   }
   return false;
}
```

Problem 7—Reading data structures from files (15 points)

```
/*
 * File: ElectionData.js
 * ---------------------
 * This file implements the ElectionData class.
 */

/*
 * Creates a new ElectionData object and initializes it from the
 * specified data file.  The data file consists of entries, one
 * for each constituency, separated by blank lines.  Each entry
 * starts with the name of the constituency, followed by any
 * number of lines of the form
 *
 *     Candidate Name (Party) votes
 *
 * If the file does not exist, the ElectionData factory method
 * returns null.
 */

function ElectionData(filename) {
   var lines = File.readLines(filename);
   if (lines === undefined) return null;
   var constituencies = [ ];
   var mapByConstituency = { };
   var line = lines.shift();
   while (line !== undefined) {
      var name = line;
      var results = [ ];
      line = lines.shift();
      while (line !== undefined && line !== "") {
         results.push(createResultEntry(line));
         line = lines.shift();
      }
      constituencies.push(name);
      mapByConstituency[name] = results;
      line = lines.shift();
   }
   return {
      getConstituencyNames: function() { return constituencies; },
      getResults: function(name) { return mapByConstituency[name]; }
   };
}

/*
 * Parses a line into a result entry for one candidate.  The result of
 * calling createResultEntry is an aggregate containing the fields
 * candidate, party, and votes.  This function does not check for
 * errors in the data file, since doing so was not required by the
 * problem specification.
 */

function createResultEntry(line) {
   var openParen = line.indexOf("(");
   var closeParen = line.indexOf(")");
   return {
     candidate: line.substring(0, openParen).trim(),
     party: line.substring(openParen + 1, closeParen).trim(),
     votes: parseInt(line.substring(closeParen + 1).trim())
   };
}
```