Eric Roberts and Jerry Cain
CS 106J

Handout #54
June 14, 2017

Answers to Final Examination

The complete histogram of percentage grades looks like this:

N = 29													
Mean = 67.0													
Median = 72						66				89		97	
		38		54		66	73			88		96	
		38		53		65	72		82	86		95	
	18	35	43	51	63	65	70	75	80	86	90	95	100

You can determine your letter grade by looking up the score in the following table:

Range	Grade	N
95–100	A+	5
84–94	A	5
77–83	A–	2
73–76	B+	2
65–72	B	6
61–64	B–	1
57–60	C+	0
47–56	C	3
43–46	C–	1
25–42	D	3
00–24	NP	1

Problem 1—Short answer (10 points)

1a) The `enigma` function implements the algorithm credited to the Greek mathematician Eratosthenes (3rd century BCE) for determining prime numbers, which is generally called the ***Sieve of Eratosthenes***. The k^{th} element of the returned array has the value `true` if k is prime. The result of calling `enigma(16)` is therefore

F	F	T	T	F	T	F	T	F	F	F	T	F	T	F	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

1b) The value produced by the outer call to `mystery` is the string `"314159"`; the quotes would not be printed in this problem but were accepted for the exam.

Problem 2—Simple graphics (15 points)

```
/*
 * Creates a GCompound consisting of a rectangular frame that contains
 * a set of GLabel objects, one for each line in the string array lines.
 */

function createTextBox(lines, font) {
   var labels = [];
   var width = 0;
   for (var i = 0; i < lines.length; i++) {
      var label = GLabel(lines[i]);
      label.setFont(font);
      if (label.getWidth() > width) width = label.getWidth();
      labels.push(label);
   }
   width += 2 * TEXT_MARGIN;
   var height = 2 * TEXT_MARGIN + labels.length * labels[0].getHeight();
   var frame = GRect(0, 0, width, height);
   var textBox = GCompound();
   textBox.add(frame);
   var y = TEXT_MARGIN + labels[0].getAscent();
   for (var i = 0; i < labels.length; i++) {
      textBox.add(labels[i], TEXT_MARGIN, y);
      y += labels[i].getHeight();
   }
   return textBox;
}
```

Problem 3—Interactive graphics (20 points)

```
/*
 * Simulates the arcade game of Whac-A-Mole in which a circle turns
 * black at random intervals and the user has to click on that circle
 * before it turns white again.  Clicking on a black circle removes it
 * from the window.
 */

function WhacAMole() {
   var gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   createCircles(gw);
   var circle = null;
   var timeStep = function() {
      var x = randomInteger(0, GWINDOW_WIDTH);
      var y = randomInteger(0, GWINDOW_HEIGHT);
      if (circle !== null) circle.setFilled(false);
      circle = gw.getElementAt(x, y);
      if (circle !== null && !circle.isFilled()) {
         circle.setFilled(true);
      }
   };
   var clickAction = function(e) {
      var obj = gw.getElementAt(e.getX(), e.getY());
      if (obj !== null && obj.isFilled()) {
         gw.remove(obj);
         circle = null;
      }
   };
   var timer = setInterval(timeStep, TIME_STEP);
   gw.addEventListener("click", clickAction);
}

/*
 * Creates the pattern of circular holes on the graphics window.  The
 * circles form a square matrix with N_CIRCLES_PER_ROW in each row
 * and column.  The diameter of each circle is given by the constant
 * CIRCLE_SIZE and the space between each circle is given by the
 * constant CIRCLE_SEP.
 */

function createCircles(gw) {
   var delta = CIRCLE_SIZE + CIRCLE_SEP;
   for (var i = 0; i < N_CIRCLES_PER_ROW; i++) {
      var y = CIRCLE_SEP / 2 + i * delta;
      for (var j = 0; j < N_CIRCLES_PER_ROW; j++) {
         var x = CIRCLE_SEP / 2 + j * delta;
         var hole = GOval(CIRCLE_SIZE, CIRCLE_SIZE);
         gw.add(hole, x, y);
      }
   }
}
```

Problem 4—Strings (15 points)

```
/*
 * Generates a random permutation suitable for use as a reflector.  The
 * conditions necessary to be a reflector are (1) that no letter maps to
 * itself and (2) the permutation is symmetric so that if A maps to B,
 * B must map to A.
 */

function generateRandomReflector() {
   var candidates = ALPHABET;
   var reflector = STARTER;
   var base = "A".charCodeAt(0);
   while (candidates.length > 0) {
      var c1 = candidates.charAt(0);
      candidates = candidates.substring(1);
      var i = randomInteger(0, candidates.length - 1);
      var c2 = candidates.charAt(i);
      candidates = candidates.substring(0, i) + candidates.substring(i + 1);
      reflector = replaceCharAt(reflector, c2.charCodeAt(0) - base, c1);
      reflector = replaceCharAt(reflector, c1.charCodeAt(0) - base, c2);
   }
   return reflector;
}

/*
 * Returns a new string in which the character at the specified index in
 * str is replaced by ch.
 */

function replaceCharAt(str, index, ch) {
   return str.substring(0, index) + ch + str.substring(index + 1);
}
```

Problem 5—Arrays (10 points)

```
/*
 * Finds and returns the first element in the array that appears
 * more than once in the array.  If no duplicated element exists,
 * findDuplicate should return the value null.
 */

function findDuplicate(array) {
   for (var i = 0; i < array.length; i++) {
      if (array.indexOf(array[i], i + 1) !== -1) {
         return array[i];
      }
   }
   return null;
}
```

If you didn't think of using **`indexOf`**, the same functionality is easy to achieve using a helper function.

Problem 6—Data structures (20 points)

```
/*
 * Prints a cheat sheet for the Adventure game showing the name of each
 * object, its short description in parentheses, and the short description
 * of its initial location.  After each object in the list, the
 * printCheatSheetForObjects function goes through the rooms data
 * structure and print out a line for each entry in which that object
 * acts as a key to a locked passage.
 */

function printCheatSheetForObjects(objects, rooms) {
   for (var objectName in objects) {
      var obj = objects[objectName];
      var desc = obj.getDescription();
      var loc = obj.getInitialLocation();
      if (loc !== "PLAYER") {
         loc = rooms[loc].getShortDescription();
      }
      console.log(objectName + " (" + desc + ") starts: " + loc);
      for (var roomName in rooms) {
         var room = rooms[roomName];
         var motionTable = room.getMotionTable();
         for (var i = 0; i < motionTable.length; i++) {
            var entry = motionTable[i];
            if (objectName === entry.getKeyName()) {
              var dir = entry.getDirection();
              var short = room.getShortDescription();
              console.log("  Needed for " + dir + " from " + short);
            }
         }
      }
   }
}
```

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

```
/* Constants */

const STECKERBOARD_PAIRS = 4;

/*
 * Reads a data file into an internal data structure for the Enigma
 * codebook.  Each line of the data file has the form
 *
 *     <date> <order> <setting> <stecker>
 *
 * where the individual components of the line have the following values:
 *
 * - <date> is the date, written as a string (without the quotes).
 * - <order> is the rotor order, written as a three-digit integer.
 * - <setting> is the rotor setting, written as three letters.
 * - <stecker> is a sequence of letter pairs separated by spaces.
 *
 * The result of calling readEnigmaCodebook is a map in which the keys
 * are dates and the values are aggregates with the fields rotorOrder,
 * rotorSetting, and steckerPairings.  The rotorOrder field is an
 * integer, the rotorSetting field is a string, and the steckerPairings
 * field is an array of two-letter strings.
 */

function readEnigmaCodebook(filename) {
   var lines = File.readLines(filename);
   var codebook = { };
   var line = lines.shift();
   while (line !== undefined) {
      var space = line.indexOf(" ");
      var date = line.substring(0, space);
      var entry = { };
      var start = space + 1;
      space = line.indexOf(" ", start);
      entry.rotorOrder = parseInt(line.substring(start, space));
      start = space + 1;
      space = line.indexOf(" ", start);
      entry.rotorSetting = line.substring(start, space);
      entry.steckerPairing = [ ];
      start = space + 1;
      for (var i = 0; i < STECKERBOARD_PAIRS; i++) {
         space = line.indexOf(" ", start);
         if (space === -1) space = line.length;
         entry.steckerPairing.push(line.substring(start, space));
         start = space + 1;
      }
      codebook[date] = entry;
      line = lines.shift();
   }
   return codebook;
}
```