Eric Roberts and Jerry Cain
CS 106J

Handout #52A
June 9, 2017

Answers to Practice Final #2

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)

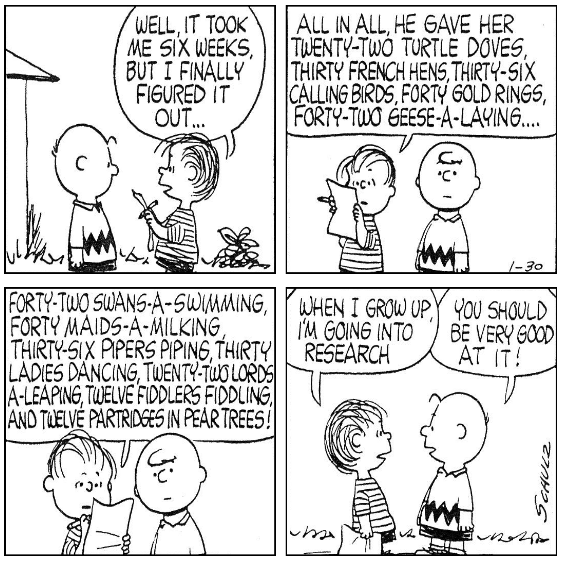
1a) **`array`**

0	12	22	30	36	40	42	42	40	36	30	22	12
0	1	2	3	4	5	6	7	8	9	10	11	12

The values for this problem (which was taken from an autumn quarter final and thus came at the right season) indicate the total number of gifts in each category if you take the words to "The Twelve Days of Christmas" literally. At the end of our true love's gift-giving spree, the total haul contains:

12 Partridges in pear trees	42 Swans-a-swimming
22 Turtle doves	40 Maids-a-milking
30 French hens	36 Ladies dancing
36 Calling birds	30 Lords-a-leaping
40 Gold rings	22 Pipers piping
42 Geese-a-laying	12 Drummers drumming

Charles M. Schulz offered a lovely rendition of this problem in 1963:


WELL, IT TOOK ME SIX WEEKS, BUT I FINALLY FIGURED IT OUT...
ALL IN ALL, HE GAVE HER TWENTY-TWO TURTLE DOVES, THIRTY FRENCH HENS, THIRTY-SIX CALLING BIRDS, FORTY GOLD RINGS, FORTY-TWO GEESE-A-LAYING....
1-30
FORTY-TWO SWANS-A-SWIMMING, FORTY MAIDS-A-MILKING, THIRTY-SIX PIPERS PIPING, THIRTY LADIES DANCING, TWENTY-TWO LORDS A-LEAPING, TWELVE FIDDLERS FIDDLING, AND TWELVE PARTRIDGES IN PEAR TREES!
WHEN I GROW UP, I'M GOING INTO RESEARCH
YOU SHOULD BE VERY GOOD AT IT!
SCHULZ

1b) The issue here is figuring out exactly which variable or field each occurrence of **`x`** refers to. The answer is the string consisting of `"" + 10 + (11 * 6)`, or `"1066"`.

Problem 2—Simple graphics (15 points)

```
/*
 * Creates a GCompound object that serves as a button.  The button is
 * composed of two semicircular GArcs that form the ends of the button,
 * two GLines that mark the top and bottom edges of the button, and
 * a GLabel that shows the button label.
 */

function createButton(str) {
   var button = GCompound();
   var label = GLabel(str);
   label.setFont(BUTTON_FONT);
   var len = label.getWidth();
   var r = BUTTON_HEIGHT / 2;
   var leftEnd = GArc(0, 0, 2 * r, 2 * r, 90, 180);
   var rightEnd = GArc(len, 0, 2 * r, 2 * r, 90, -180);
   var topLine = GLine(r, 0, r + len, 0);
   var bottomLine = GLine(r, 2 * r, r + len, 2 * r);
   button.add(leftEnd);
   button.add(rightEnd);
   button.add(topLine);
   button.add(bottomLine);
   button.add(label, r, r + BUTTON_LABEL_DY);
   return button;
}
```

Problem 3—Interactive graphics (20 points)

```
/*
 * File: GraphicNim.js
 * -------------------
 * This program plays a graphical version of the game of Nim.
 */

import "graphics";

/* Constants */

const GWINDOW_WIDTH = 496;
const GWINDOW_HEIGHT = 75;
const N_COINS = 11;
const COIN_SIZE = 32;
const COIN_FILL_COLOR = "LightGray";

/* Derived constants */

const COIN_SEP = (GWINDOW_WIDTH - N_COINS * COIN_SIZE) / (N_COINS + 1);

/* Main program */

function GraphicNim() {
   var gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   var coins = createCoinArray(gw);
   var clickAction = function(e) {
      var coin = gw.getElementAt(e.getX(), e.getY());
      if (coin !== null) {
         var index = coins.indexOf(coin);
         var coinsToRemove = coins.length - index;
         if (coinsToRemove <= 3) {
            for (var i = 0; i < coinsToRemove; i++) {
               gw.remove(coins.pop());
            }
         }
      }
   };
   gw.addEventListener("click", clickAction);
}

/*
 * Creates the initial array of coins and places them in a line across
 * the graphics window.  This function returns the array of coins.
 */

function createCoinArray(gw) {
   var array = [ ];
   var y = (GWINDOW_HEIGHT - COIN_SIZE) / 2;
   for (var i = 0; i < N_COINS; i++) {
      var x = (i + 1) * COIN_SEP + i * COIN_SIZE;
      var coin = GOval(COIN_SIZE, COIN_SIZE);
      coin.setFilled(true);
      coin.setFillColor(COIN_FILL_COLOR);
      gw.add(coin, x, y);
      array.push(coin);
   }
   return array;
}
```

Problem 4—Strings (15 points)

```
/*
 * Creates a table of contents entry in which the chapter title and the
 * page number are separated by a leader composed of alternating spaces
 * and periods.
 */

function createTocEntry(title, page) {
   var entry = title + " ";
   if (title.length % 2 === 0) entry += " ";
   var pageString = " " + page;
   var gap = TOC_LINE_LENGTH - entry.length - pageString.length;
   for (var i = 0; i < gap; i++) {
      entry += (i % 2 === 0) ? "." : " ";
   }
   entry += pageString;
   return entry;
}
```

Problem 5—Arrays (10 points)

```
/*
 * Rotates the elements of the array leftward k positions.  Elements that
 * are shifted out of the array reappear at the end.
 */

function rotateArray(array, k) {
   for (var i = 0; i < k; i++) {
      array.push(array.shift());
   }
}
```

Problem 6—Working with data structures (15 points)

```
/*
 * Returns the refund if someone tried to sell nShares of Facebook stock
 * at timeOrdered on the specified date, but Morgan Stanley did not
 * complete the sale until timeExecuted.
 */

function facebookRefund(nShares, date, timeOrdered, timeExecuted) {
   var priceOrdered = lookupSharePrice(date, timeOrdered);
   var priceExecuted = lookupSharePrice(date, timeExecuted);
   var refund = nShares * (priceOrdered - priceExecuted);
   if (refund < 0) refund = 0;
   return refund;
}

/*
 * Looks up the Facebook share price for the specified date and time.
 */

function lookupSharePrice(date, time) {
   for (var i = 0; i < FB_SHARE_PRICE_DATA.length; i++) {
      var entry = FB_SHARE_PRICE_DATA[i];
      if (entry.date === date && entry.time === time) {
         return entry.price;
      }
   }
   alert("No record for " + data + " " + time);
}
```

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

```
/*
 * Reads files from the specified directory into the database of letters.
 * The file names in the directory are assumed to be consecutive integers
 * beginning with 1, up to the number of letters in the directory.  The
 * return value is an array of letters in which the element at the index
 * given by the file name is an aggregate consisting of header fields and
 * the special field body, which contains an array of the lines in the
 * message body.  The entry at index 0 is set to null so that the indices
 * and file names match.
 */

function readLetters(dir) {
   var letters = [ null ];
   var index = 1;
   var lines = File.readLines(dir + "/" + index);
   while (lines !== undefined) {
      letters.push(parseLetter(lines));
      index++;
      lines = File.readLines(dir + "/" + index);
   }
   return letters;
}

/*
 * Parses an array of lines into the internal data structure for a
 * letter.  The letter begins with a set of headers, each of which
 * consists of a key, a colon, and the value of the header.  The
 * headers are followed by a blank line and then by the body of the
 * message.
 */

function parseLetter(lines) {
   var letter = { };
   var line = lines.shift();
   while (line !== "") {
      var colon = line.indexOf(":");
      if (colon === -1) alert("Missing colon in header");
      var key = line.substring(0, colon).trim();
      var value = line.substring(colon + 1).trim();
      letter[key] = value;
      line = lines.shift();
   }
   letter.body = lines;
   return letter;
}
```