Jerry Cain
CS106AX

Handout #15S
October 29th, 2026

Midterm Examination Solution

Wonder CA Ben alongside Wonder SLs Sabrina and Eugene spent their Sunday afternoon grading all of the midterms. I'm happy to report that they've been fully graded, and your results will soon be published via Gradescope at https://www.gradescope.com.

The exam was challenging, but many of you still did brilliantly, and as a group you exceeded expectations. I'm happy to go with my traditional curve for an accelerated course where I set the median grade to map to the A/A-. The median grade was a 52.0 out of 70 with a fairly wide standard deviation of 16.0.

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

Median = 52.0

Range	Grade	N
65–70	A+	2
56–64.5	A	8
50–55.5	A–	4
43–49.5	B+	2
31–42.5	B	2
26–30.5	B–	2
18–22.5	C+	1
10–17.5	C	2

Solution 1: Simple JavaScript (10 points)

(1a) [3 points] Compute the value of each of the following JavaScript expressions:

`10 ** 2 - 71 % 7`	99
`3 * 3 < 8 && 1 / 0 > 1000`	**`false`**
`7 * 7 + "ERS" + 4 + 9`	**`"49ERS49"`**

(1b) [3 points] Suppose that the function **`bamboozle`** is defined as follows:

```
function bamboozle() {
   let numbers = [1];
   for (let i = 0; i < 7; i++) {
      for (let j = 0; j < numbers.length; j++) {
         numbers[j] *= 2;
      }
      numbers.push(1);
   }
   return numbers;
}
```

If you look at the code, you'll see that the function pushes a new value onto the array for every value of **`i`** between 0 and 6, which means that the array will eventually contain 8 elements with indices ranging from 0 to 7. Work through the function carefully and indicate the value of each of the elements in the array that **`bamboozle`** returns:

0	1	2	3	4	5	6	7
`128`	**`64`**	**`32`**	**`16`**	**`8`**	**`4`**	**`2`**	**`1`**

Why? Each 1 that gets pushed is doubled once for every additional pass.

(1c) [4 points] What output is produced by a call to `haunted()`?

```
function haunted() {
   let ghost = "poltergeist";
   let witch = "cauldron";

   let spell = function(word, n) {
      let potion = word.substring(n - 2, n + 2);
      return potion.charAt(1) + witch.substring(0, n);
   };

   let broom = witch.lastIndexOf("o");
   witch = "sorceress";
   let bat = spook(spell, ghost, broom);
   console.log(bat);
}

function spook(magic, word, index) {
   let result = magic(word, index + 1);
   let offset = "zombie".charCodeAt(0) - "witch".charCodeAt(0);
   return result.substring(1) + offset;
}
```

Output: **`sorcere3`**

Solution 2: Graphics, Animation, and Balloons (15 points)

Implement a graphical program in which balloons float up through the graphics window and the user pops them by clicking.

- Every 800 milliseconds, a new balloon is placed just below the bottom of the graphics window at a random x coordinate that allows the full balloon to ultimately appear on screen.
- All balloons are filled, randomly colored ovals, **`BALLOON_WIDTH`** pixels wide and **`BALLOON_HEIGHT`** pixels high.
- Each balloon rises 1 pixel every 20 milliseconds.
- Clicking a balloon pops it, and it disappears from the window. If balloons overlap, the click pops just the topmost one. Clicks that miss every balloon have no effect.
- A balloon that floats completely off the top of the window is removed.

```
const GWINDOW_WIDTH = 400;
const GWINDOW_HEIGHT = 300;
const BALLOON_WIDTH = 30;
const BALLOON_HEIGHT = 40;
const RISE_SPEED = 1;
const RISE_DELAY = 20;
const LAUNCH_DELAY = 800;
```

A few things:

- A balloon's animation must stop as soon as the balloon is popped or has floated above the screen. No animation should keep running for a balloon that's gone.
- A balloon's timer is known only inside the function that launched it, so the click handler can't easily call **`clearInterval`**. Instead, the balloon's own animation should notice the balloon was popped and stop itself. The lesser-known **`setVisible`** and **`isVisible`** methods on **`GObject`**s may help. Their prototypes are what you'd expect:

```
void setVisible(isVisible);
bool isVisible();
```

Present your implementation on the next page.

(space for the answer to problem #2 appears on the next page)

```
/*
 * File: BalloonPop.js
 * -------------------
 * This program launches balloons that float up the graphics window and
 * lets the user pop them by clicking.
 */

/* Constants (in pixels) */
const GWINDOW_WIDTH = 400;
const GWINDOW_HEIGHT = 300;
const BALLOON_WIDTH = 30;
const BALLOON_HEIGHT = 40;
const RISE_SPEED = 1;

/* Constants (in milliseconds) */
const RISE_DELAY = 20;
const LAUNCH_DELAY = 800;

function BalloonPop() {
   let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   setInterval(function() {
      launchBalloon(gw);
   }, LAUNCH_DELAY);
   gw.addEventListener("click", function(e) {
      let balloon = gw.getElementAt(e.getX(), e.getY());
      if (balloon === null) return;
      balloon.setVisible(false);
   });
}

function launchBalloon(gw) {
   let ulx = randomInteger(0, GWINDOW_WIDTH - BALLOON_WIDTH);
   let balloon = GOval(ulx, GWINDOW_HEIGHT, BALLOON_WIDTH, BALLOON_HEIGHT);
   balloon.setFilled(true);
   balloon.setFillColor(randomColor());
   gw.add(balloon);

   let timer = setInterval(function() {
      if (!balloon.isVisible() || balloon.getY() + BALLOON_HEIGHT <= 0) {
         clearInterval(timer);
         gw.remove(balloon);
      } else {
         balloon.move(0, -RISE_SPEED);
      }
   }, RISE_DELAY);
}
```

Solution 3: Strings and Autokey Ciphers (15 points)

The **autokey cipher** shifts each letter by a different amount, taken from a key that grows out of the message itself. Given a lowercase **`word`** and a lowercase **`primer`**, the key is **`primer + word`**. Each **`word[i]`** is shifted forward by the key letter at index **`i`**, where **`'a'`** shifts by 0, **`'b'`** by 1, …, **`'z'`** by 25. A shift that runs past **`'z'`** wraps around to **`'a'`**.

As an example, **`encrypt("buffalo", "oxen")`** returns **`"prjsbft"`**, and here's why:

- The key is **`"oxen" + "buffalo"`**—that is, **`"oxenbuffalo"`**. Only its first seven letters, **`"oxenbuf"`**, end up mattering.
- **`'b'`** shifted by **`'o'`** (14) gives **`'p'`**.
- **`'u'`** shifted by **`'x'`** (23) runs past **`'z'`** and wraps around to **`'r'`**.
- **`'f'`** shifted by **`'e'`** (4) gives **`'j'`**.
- **`'f'`** shifted by **`'n'`** (13) gives **`'s'`**.
- **`'a'`** shifted by **`'b'`** (1) gives **`'b'`**.
- **`'l'`** shifted by **`'u'`** (20) wraps around to **`'f'`**.
- **`'o'`** shifted by **`'f'`** (5) gives **`'t'`**.

Use the next page to implement the **`encrypt`** function. You may assume both strings are nonempty and contain only lowercase letters. Your implementation must rely on **`charCodeAt`** and **`String.fromCharCode`** to manage each character shift.

(space for the answer to problem #3 appears on the next page)

```
/*
 * Function: encrypt
 * -----------------
 * Returns the autokey encryption of word, where each letter is shifted
 * forward by the matching letter of primer + word ('a' shifts by 0,
 * 'z' by 25), wrapping around past 'z'.
 */
function encrypt(word, primer) {
   let key = primer + word;
   let result = "";
   for (let i = 0; i < word.length; i++) {
      let shift = key.charCodeAt(i) - "a".charCodeAt(0);
      let code = word.charCodeAt(i) + shift;
      if (code > "z".charCodeAt(0)) code -= 26; // % 26 works as well
      result += String.fromCharCode(code);
   }
   return result;
}
```

Solution 4: Arrays and Soulmate Numbers (15 points)

The first two **soulmate numbers** are 1 and 2. After that, each soulmate number is the smallest integer greater than the previous one that can be written as the sum of two **different** soulmate numbers **in precisely one way**.

The metaphor here? Each new soulmate number has exactly one pair of previously generated numbers that were meant for each other. Everyone needs a soulmate, people.

Here's how the sequence continues post 1 and 2:

- 3 = 1 + 2. That's the only way, so 3 is in.
- 4 = 1 + 3. Again, that's the only way, so 4 is in. 2 + 2 doesn't count, since the two numbers must be different.
- 5 = 1 + 4 = 2 + 3. That's two ways, so 5 is out.
- 6 = 2 + 4. That's the only way, so 6 has soul and is in.
- 7 = 1 + 6 = 3 + 4. That's two ways, so 7 doesn't belong.
- 8 = 2 + 6 is the only way, so 8 gets included.

The sequence continues 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, etc. Nobody has found a single function to generate all of them, so the only way to know whether a number is a soulmate number is to brute-force generate the list and see if it's in there.

For this problem, you're to implement `soulmate(n)` to return an array of the first `n` soulmate numbers in increasing order, and you're to use the space on the next page to do so. You should assume the supplied `n` is 2 or greater.

(space for the answer to problem #4 appears on the next page)

```
/*
 * Function: soulmate
 * ------------------
 * Returns an array of the first n soulmate numbers, where each term
 * after 1 and 2 is the smallest integer larger than the previous
 * term that is the sum of two distinct earlier terms in exactly
 * one way.
 */
function soulmate(n) {
   let terms = [1, 2];
   let candidate = 3;
   while (terms.length < n) {
      let count = 0;
      for (let i = 0; i < terms.length; i++) {
         for (let j = i + 1; j < terms.length; j++) {
            if (terms[i] + terms[j] === candidate) count++;
         }
      }
      if (count === 1) terms.push(candidate);
      candidate++;
   }
   return terms;
}
```

Solution 5: Data Structures and Rock, Paper, Scissors++ (15 points)

To cut down on ties in rock-paper-scissors, two peeps by the names of Sam Kass and Karen Bryla invented a five-move version, which the American television show *The Big Bang Theory* ultimately made famous. Each move beats exactly two others:

```
const PREVAILS = {
   rock: ["scissors", "lizard"],
   paper: ["rock", "spock"],
   scissors: ["paper", "lizard"],
   lizard: ["spock", "paper"],
   spock: ["scissors", "rock"]
};
```

You really don't need to know either the original game or this new one. **`PREVAILS`** tells you everything about who wins.

A tournament is recorded as an array of rounds. Each round lists the two players and their throws, in the same order:

```
let tournament = [
   { players: ["lea", "pooja"], throws: ["rock", "spock"] },
   { players: ["nick", "claire"], throws: ["lizard", "paper"] },
   { players: ["pooja", "nick"], throws: ["scissors", "scissors"] },
   { players: ["claire", "lea"], throws: ["spock", "lizard"] },
   { players: ["pooja", "claire"], throws: ["paper", "scissors"] },
   { players: ["lea", "nick"], throws: ["scissors", "paper"] },
   { players: ["nick", "pooja"], throws: ["rock", "lizard"] }
];
```

Go ahead and implement the **`scoreboard`** function to return an object mapping each player's name to a record of their wins, losses, and ties. If both players throw the same move, the round is a tie for both of them. And if each of the two players throws something different, **`PREVAILS`** makes it clear which of the two wins.

For the tournament above, a call to **`scoreboard(tournament)`** returns:

```
{
   lea: { wins: 2, losses: 1, ties: 0 },
   pooja: { wins: 1, losses: 2, ties: 1 },
   nick: { wins: 2, losses: 1, ties: 1 },
   claire: { wins: 1, losses: 2, ties: 0 }
}
```

Use the space on the next page to present your implementation.

(space for the answer to problem #5 appears on the next page)

```
/*
 * Function: scoreboard
 * --------------------
 * Returns an object mapping each player in tournament to
 * an object recording that player's wins, losses, and ties.
 */
function scoreboard(tournament) {
   let board = {};
   for (let round of tournament) {
      for (let name of round.players) {
          if (board[name] === undefined) {
             board[name] = { wins: 0, losses: 0, ties: 0 };
          }
      }
      let a = round.players[0];
      let b = round.players[1];
      let throwA = round.throws[0];
      let throwB = round.throws[1];
      if (throwA === throwB) {
         board[a].ties++;
         board[b].ties++;
      } else if (PREVAILS[throwA].indexOf(throwB) !== -1) {
         board[a].wins++;
         board[b].losses++;
      } else {
         board[b].wins++;
         board[a].losses++;
      }
   }
   return board;
}
```