Midterm Examination

Name (please print) ____________________________________________
Section Leader ______________________________________________

General instructions

Answer each of the five questions included in the exam. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit.

Each question is marked with the number of points assigned to that problem. The total number of points is 70. We intend for the number of points to be roughly comparable to the number of minutes you should spend on that problem. This leaves you with an additional 50 minutes to check your work or recover from false starts.

In all questions, you may include methods or definitions that have been developed in the course, either by writing the import line for the appropriate Karel package, or by giving the name of the method and the handout or chapter number in which that definition appears.

Unless otherwise indicated as part of the instructions for a specific problem, comments will not be required on the exam. Uncommented code that gets the job done will be sufficient for full credit on the problem. On the other hand, comments may help you to get partial credit if they help us determine what you were trying to do.

The examination is open-book, and you may make use of any texts, handouts, or course notes. You may not, however, use a computer of any kind.

THE STANFORD UNIVERSITY HONOR CODE

A. The Honor Code is an undertaking of the students, individually and collectively:
   (1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
   (2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

(signed)
Problem 1: Karel the Robot (10 points)

In this problem, you job is to program Karel to create a calendar by putting down beepers in a pattern that corresponds to the days in a particular month. Karel begins the task in the upper left row of a $7 \times 6$ world. On one of the intersections on that row—corresponding to the first day of the month in question—there is a beeper pile containing exactly the same number of beepers as there are days in the month. For example, for a February that begins on a Monday and has 28 days, the initial state of the world would look like this:

What Karel needs to do is:

1. Walk across the top row to find the beeper pile.
2. Pick up all the beepers.
3. Put the beepers down, one at a time, starting at the intersection on which it found the pile and then continuing across each row in turn until it runs out of beepers.

Thus, given the starting configuration for February, Karel should finish with a world diagram that looks like this:
Karel may count on the following facts about the world:

- Karel’s world always has seven avenues and six streets.
- Karel begins at the corner of 6th Street and 1st Avenue, facing east, with an empty beeper bag.
- There is a pile with the correct number of beepers somewhere on 6th Street.
- At the end of execution, Karel should be positioned on top of the beeper representing the last day of the month, facing east.

Use the rest of this page and the next to present your implementation. You are strongly encouraged to decompose, writing helper functions and referencing existing ones that appear anywhere the course reader or in any course handout.

(space for the answer to problem #1 also appears on the next page)
Answer to Problem 1 (continued):
Problem 2: Simple Java expressions, statements, and methods (10 points)

(2a) Compute the value of each of the following JavaScript expressions:

\[
\begin{align*}
5 + 7 \times 7 - 2 + 1 + 5 \% 6 + 6 \times 4 \% 9 & \\
"C" == "CC" || "Y" < "X" + "Y" & \\
7 + 1 + "1" + 4 \times 7 & 
\end{align*}
\]

(2b) Assume that the method `mystery` has been defined as follows:

```javascript
function mystery(word) {
    let result = "";
    while (word.length > 1) {
        if (word.charAt(0) > word.charAt(1)) {
            word = word.substring(0, Math.floor(word.length/2));
        } else {
            word = word.substring(Math.floor(word.length/2));
        }
        result += word;
    }
    return result;
}
```

What is the value returned by `mystery("xylophones")`?

Answer to problem 2b:
What output is printed by the following `Problem2c` program?

```javascript
function Problem2c() {
    let s = "this is a test";
    let fn = function(a, b, c) {
        return s.substring(a, b) + c.substring(a, b);
    };
    s = outperform(fn, s.indexOf("is"), s.lastIndexOf("si"));
    s.toUpperCase();
    console.log(s);
}

function outperform(f, x, y) {
    let str = f(y + 1, 2 * x, "good luck");
    str += str.charAt(3);
    return str.toLowerCase();
}
```

Answer to problem 2c:
Problem 3: Simple JavaScript programs (15 points)

When creditors like Visa, MasterCard, and American Express issue new credit cards, they ensure the new account numbers are valid according to Luhn’s algorithm. Luhn’s algorithm can then be used by online retailers to instantly verify credit card information as possibly valid and immediately reject most mistyped or intentionally manufactured numbers.

Luhn’s algorithm is a digit manipulation algorithm that works like this:

- Isolate every digit, starting from the right and moving left, doubling every second one. When this doubling produces a value greater than 9, subtract 9 from it. For example, 596825 would produce:
  - 5 on behalf of the 5 in the ones place
  - 2 * 2 = 4 on behalf of the 2 in the tens place
  - 8 on behalf of the 8 in the hundreds place
  - 2 * 6 - 9 = 3 on behalf of the 6 in the thousands place
  - 9 on behalf of the 9 in the ten thousands place
  - 2 * 5 - 9 = 1 on behalf of the 5 in the hundred thousands place

- Sum all of the transformed digits to produce the Luhn digit sum. For example, the 596825 above has a Luhn digit sum of 5 + 4 + 8 + 3 + 9 + 1 = 30.
- If the Luhn digit sum ends in a 0, then and only then is the original number valid according to Luhn’s algorithm. For example, 596825 is technically a valid credit card number according to Luhn’s algorithm, whereas 596725 (where there’s a 7 in place of the 8) is not.

Using the space on the next page, write a program called Luhn that accepts a positive integer and returns true if the provided number is valid according to Luhn’s algorithm, and false otherwise.

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

```javascript
function Luhn(number) {
```
Problem 4: Using graphics and animation (20 points)

Write a program that allows the user to place randomly shaped quadrilaterals in a 300px by 500px graphics window. The program is designed to operate as follows:

- Initiate the construction of a quadrilateral—as a `GPolygon`—by clicking the mouse anywhere in the graphics window. A small, solid black circle of diameter `DOT_DIAMETER` should be drawn so that its center overlays the x and y coordinates of the mouse click.
- The second, third, and fourth mouse clicks should introduce the second, third, and fourth vertices of the quadrilateral, and each click should leave a small, solid black circle just as one was left by the first click. The fourth click should effectively close off the quadrilateral by adding it to the graphics window as a filled `GPolygon` with a black border and a random fill color.
- Additional polygons are introduced to the graphics window in exactly the same manner: the first single click initiates polygon construction; the second and third click define the next two vertices, and the fourth click defines the fourth and final vertex, at which point the implied quadrilateral is added to the graphics window, outlined in black, and filled with a random color.
- As each quadrilateral is introduced to the graphics window, each should be configured to change fill colors every five seconds from the time it was added. The color changes never stop.
- Quadrilaterals can be shaped and placed anywhere in the graphics window, even if their placement eclipses previously placed ones.

So, after 10 mouse clicks, the graphics application might look like the image below on the left, and after 25 clicks, the same graphics application might look like the image below on the right:

![Image showing 10 mouse clicks](image1.png) ![Image showing 25 mouse clicks](image2.png)

Use the next page to present your entire implementation of your `Quadrilaterals` program.

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

/* Constants (in pixels) */
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 300;
const DOT_DIAMETER = 4;

/* Main program */
function Quadrilaterals() {

Problem 5: Strings (15 points)

Pig Latin is just one example of a language game designed to render spoken words to be incomprehensible to the untrained ear. **B-Language** is another such language game—known primarily in Germany—where words are transformed such that every vowel cluster is reduplicated with a leading "b". So, the B-Language translation of "Deutsche Sprachen"—German for "German Language"—is "Deubeutsche Sprabachebe". In particular, notice the "eu" of "Deutsche" became the "eubeu" of "Deubeutschebe". Restated, "eu" was reduplicated with a leading "b".

Here are some examples of how `translate`, which you’ll implement to translate other English (not German) words comprised solely of lowercase letters, should work:

\[
\begin{align*}
\text{translate("quick")} & \Rightarrow "quibuick" \\
\text{translate("spaghetti")} & \Rightarrow "spabaghebettibi" \\
\text{translate("adieu")} & \Rightarrow "abadieubieu" \\
\text{translate("audiophile")} & \Rightarrow "aubaudiobiophibilebe" \\
\text{translate("queueing")} & \Rightarrow "queueibueueing"
\end{align*}
\]

Use the next page to present your implementation of `translate`, which accepts a word and returns its B-Language translation. *(Hint: monitor the transitions between vowels and other characters in much the same manner that the Pig Latin program covered in Chapter 7 monitors the transition between letters and other characters.)*

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

```javascript
function translate(word) {
```