Eric Roberts and Jerry Cain
CS 106J

Handout #27
May 5, 2017

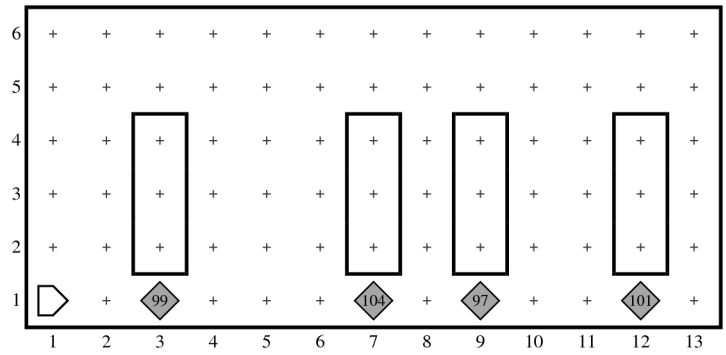
Practice Midterm Examination #2

Review session: **Today (Friday, May 5), 4:00–6:00 P.M., Gates 403**
Midterm exams: **Tuesday, May 9, 3:30–5:30 P.M., 200-030**
Tuesday, May 9, 7:00–9:00 P.M., 380-380Y

Problem 1: Karel the Robot (10 points)

> *[We will] wield technology's wonders to raise health care's quality and lower its cost.*
>
> —President Barack Obama, January 20, 2009

Given that nothing could possibly be more representative of "technology's wonders" than Karel the Robot, it is clear that a central component of ObamaCare—while it still exists—must involve putting our tireless robot to work in support of patient care. As part of meeting that challenge, you have been asked to write a program that allows Karel to monitor the temperatures of patients on a hospital ward that looks something like this:

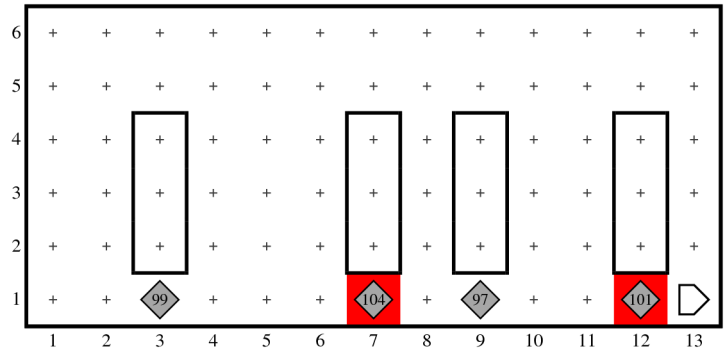


The stacks of beepers under each bed record the current temperature of the patient. Thus, the patient occupying the bed on 3rd Avenue has a temperature of 99˚, which is indicated by a stack containing 99 beepers. Although 99˚ is slightly higher than the normal body temperature of 98.6˚F, it is nothing to worry about. The patient in the 7th Avenue bed, on the other hand, has a temperature of 104˚, which is dangerously high. The 97˚ for the patient in the 9th Avenue bed is below normal, but the 101˚ temperature of the patient in the bed on 12th Avenue is again high enough to cause concern.

Your job in this problem is to implement a function called **`KarelCare`** in which Karel checks each of the patients in turn and flags any temperature that is greater than 100. One way to flag an out-of-range temperature is to paint the corner red, which Karel can do by calling the following function from the **`"extensions"`** library:

```
paintCorner("Red");
```

At the end of the program's operation, Karel's world should look like this:



In this world, the temperature values 104 and 101 are marked by a red corner.

In writing this program, you should keep the following points in mind:

1. The only part of Karel's world you need to consider is 1st Street, which is the row at the bottom of the window (the walls marking the beds are merely decorative). Karel can find where the beds are simply by looking for the stacks of beepers.
2. Karel always begins at the corner of 1st Street and 1st Avenue, facing east, with an empty beeper bag. Karel should finish on the easternmost corner of 1st Street.
3. The world can be of any length, and there can be any number of beds. The beds, moreover, can be separated by any number of open spaces, and can even be adjacent.
4. Once Karel figures out that a particular temperature needs to be flagged, it must put all the beepers back to ensure that the beeper pile continues to show the correct temperature. Given that Karel has no way of telling how many beepers are on a corner without taking them away, the number of beepers in each pile will change as the program runs. (It may help to recall that Karel starts out with an empty beeper bag.)
5. In order to meet the hospital's threshold of concern, a temperature must be strictly greater than 100. Thus, Karel should not paint the square if the patient's temperature is exactly 100, but should do so if the temperature is 101.
6. Remember that Karel has no variables and no arithmetic operations. Even so, Karel can, for example, repeat an operation 100 times by using the **`repeat`** statement.

Problem 2: Simple JavaScript expressions, statements, and methods (10 points)

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

```
10 * 9 + 8 * 7 * 6 * 5 + 4 * 3 / 2 / 1
```

_____

```
var x = 7;
(x !== 6) || (x !== 7)
```

_____

```
"E".charCodeAt(0) - "A".charCodeAt(0)
```

_____

(2b) Assume that the method **`mystery`** has been defined as given below:

```
function mystery(n) {
   while (n >= 10) {
      var k = 0;
      while (n > 0) {
         k += n % 10;
         n = Math.floor(n / 10);
      }
      n = k;
   }
   return n;
}
```

What is the value of **`mystery(1729)`**?

(2c) What output is printed by the following program:

```
/*
 * File: Problem2c.js
 * ------------------
 * This program doesn't do anything useful and exists only to
 * test your understanding of parameters and string methods.
 */

function problem2c() {
   var s1 = "Heart";
   var s2 = valentine("candy", s1);
   console.log("s1 = " + s1);
   console.log("s2 = " + s2);
}

function valentine(s1, s2) {
   var num = (s1.substring(1, 2)).length;
   s1 = s2.substring(num);
   s2 = cupid(s1, s2.charAt(0));
   return s2;
}

function cupid(s1, ch) {
   return s1 + ch.toLowerCase();
}
```

Problem 3: Simple JavaScript programs (15 points)

> *Heads. . . .*
> *Heads. . . .*
> *Heads. . . .*
> *A weaker man might be moved to re-examine his faith, if in nothing else at least in the law of probability.*
>
> —Tom Stoppard, *Rosencrantz and Guildenstern Are Dead,* 1967

Write a function **`consecutiveHeads(numberNeeded)`** that simulates tossing a coin repeatedly until the specified number of heads appear consecutively. At that point, your program should display a line on the console that indicates how many coin tosses were needed to complete the process. The following console log shows one possible execution of the program:

JavaScript Console

```
> consecutiveHeads(3);
Tails
Tails
Heads
Tails
Tails
Heads
Heads
Heads
It took 8 tosses to get 3 consecutive heads.
>
```

Don't worry about grammatical correctness. It is perfectly fine for a call to **`consecutiveHeads(1)`** to display the message "It took 1 tosses to get 1 consecutive heads."

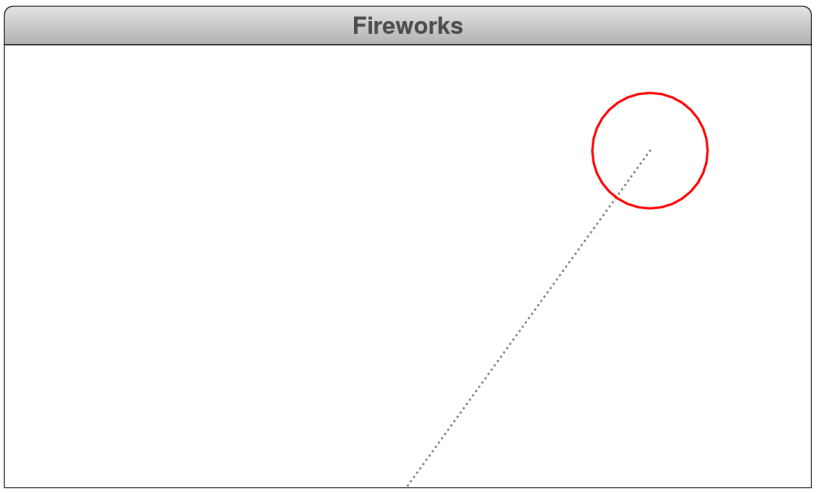
Problem 4: Using graphics and animation (20 points)

When Marissa Mayer (now CEO of Yahoo! Inc.) took CS 106A, her entry in the Graphics Contest was a screensaver program that simulated a fireworks show. Your task in this problem is to implement an exam-sized subset of her contest-sized application.

Your program should execute the following steps:

1. Create a tiny dot (an unfilled circle whose width and height are both one pixel) at the point that lies at the center of the bottom of the window and color it using a randomly chosen color.
2. Choose a random point somewhere in the top half of the window.
3. Animate the motion of the dot so that it moves to the point you chose in step 2. The dot should move so that gets to its destination in `FLIGHT_TIME` milliseconds.
4. Once the dot reaches its destination, you should change the behavior of the animation so that the circle radius increases by `DELTA_RADIUS` pixels for `EXPANSION_TIME` milliseconds.

An animation with random behavior is difficult to represent on the printed page, but the final image on the window will look something like this (the dotted line shows the flight path of the dot, although this line would not appear on the window):



Problem 5: Strings (15 points)

A ***spoonerism*** is a phrase in which the leading consonant strings of the first and last words are inadvertently swapped, generally with comic effect. Some examples of spoonerisms include the following phrases and their spoonerized counterparts (the consonant strings that get swapped are underlined):

crushing blow → blushing crow
sons of toil → tons of soil
pack of lies → lack of pies
jelly beans → belly jeans
flutter by → butter fly

In this problem, your job is to write a function

```
function spoonerize(phrase)
```

that takes a multiword phrase as its argument and returns its spoonerized equivalent. For example, you should be able to use your function to duplicate the following console session in which all the examples come from Shel Silverstein's spoonerism-filled children's book *Runny Babbit:*

JavaScript Console

```
> spoonerize("bunny rabbit")
runny babbit
> spoonerize("silly book")
billy sook
> spoonerize("take a shower")
shake a tower
> spoonerize("wash the dishes")
dash the wishes
>
```

In this problem, you are not responsible for any error-checking. You may assume that the phrase passed to **`spoonerize`** contains nothing but lowercase letters along with spaces to separate the words. You may also assume that the phrase contains at least two words, that there are no extra spaces, and that each word contains at least one vowel. What your method needs to do is extract the initial consonant substrings from the first and last words and then exchange those strings, leaving the rest of the phrase alone.

Hint: Remember that you can use methods from the book. The **`findFirstVowel`** and **`isEnglishVowel`** methods from the Pig Latin program will certainly come in handy.