CS5001 Assignment 3

Programming Language: BSL

Due Date Wednesday 1/31 at 10:00pm

Purpose To design functions for itemization data.

For this assignment and all future assignments you must upload a .rkt file in the specified language to the Handin server (handins.ccs.neu.edu).

Write enough check-expects for these exercises so that when you hit "run" no black text appears. Make sure to follow the Design Recipe!

Exercise 1 Below is a data definition for a class of shapes

```
;; Shape is one of:
;; -- Circle
;; -- Square
;; -- Rectangle

(define-struct circl (x y r outline c))
;; A Circle is a (make-circl Number Number Number Boolean Symbol)
;; interpretation: x and y determine the center of the circle,
;;   r the radius, outline whether it's outlined or solid,
;;   and c its color

(define-struct squar (x y size outline c))
;; A Square is a (make-squar Number Number Number Boolean Symbol)
;; interpretation: Supply a good interpretation of Square.

(define-struct recta (x y width height outline c))
;; A Rectangle is a (make-recta Number Number Number Number Boolean Symbol)
;; interpretation: Supply a good interpretation of Rectangle.
```

a. Add an interpretation for the `Square` and `Rectangle` classes. Both represent shapes whose borders are parallel to the borders of a canvas (window).

b. Develop the template for functions that consume `Shape`s.

c. Use the template to design `shape-shift-x`. The function consumes a `Shape`, *`sh`*, and a number, *`delta`*. It produces a `Shape` that is like *`sh`* but shifted by *`delta`* pixels along the x-axis.

d. Use the template to design `in-shape?`. The function consumes a `Shape`, *`sh`*, and a `Posn`, *`p`*, and determines whether *`p`* is inside (or on the boundary) of *`sh`*.
Domain Knowledge: for a point to be within a circle, its distance to the center must be less than or equal to the circle's radius. For a point to be within a rectangle, its x coordinate must

be between the x coordinate of the left line and the x coordinate of the right line. How do you compute the x coordinates of these lines? Naturally something analogous must hold for the y coordinates. Remember that squares are just special rectangles.

e. Use the template to design `shape-draw`. The function consumes a `Shape`, *`sh`* and a `Scene`, *`sc`* and adds *`sh`* to *`sc`*.

Exercise 2

A *Finite State Machine* is an abstract (and general) encoding of a fairly common scenario: it represents the idea of a finite collection of states, and a set of allowable transitions between them. For example, the traffic-light animation used three states (green, yellow, or red), and three transitions (green -> yellow, yellow -> red, red -> green) that occurred once per tick. As another example, a telephone might be in one of three states (idle, ringing, or in-use), and might have several transitions: idle -> ringing when a call comes in, ringing -> idle if you choose to ignore the call, ringing -> in-use if you answer the phone, and in-use -> idle when you hang up. In general, there could be an arbitrary number of states, and arbitrary transitions between the states.

In this problem, you'll *design* a world program that implements a Finite State Machine, that recognizes when a user types `"good"`, `"goood"`, `"goooood"`, or similar variants with even more `"o"`s in them. These letters must appear consecutively: if any other characters appear in the middle (like in `"goCS2500od"`), your program should not accept the input. However, it doesn't matter how many letters appear before or after: `"CS2500 is gooooooood!"`would be accepted.

Concretely:

- Your program will have five states:
    - `START`: haven't seen any part of `"good"` yet
    - `G`: have seen the intial `"g"`
    - `O1`: have seen at least one `"o"`
    - `O2`: have seen at least two `"o"`s
    - `D`: have seen the final `"d"`
- From state `START`, if the user types a `"g"`, move to state `G`. If the user types anything else, stay in `START`.
- From state `G`, if the user types an `"o"`, move to state `O1`. If the user types a `"g"`, stay in state `G`. If the user types anything else, go back to state `START`.
- From state `O1`, if the user types a second `"o"`, go to state `O2`. If the user types a `"g"`, go back to state `G`. If the user types anything else, go back to state `START`.
- From state `O2`, if the user types another `"o"`, stay in state `O2`. If the user types a `"g"`, go back to state `G`. If the user types a `"d"`, go to state `D`. If the user types anything else, go back to state `START`.
- From state `D`, no matter what the user types, stay in state `D`.

- To render your program: if the user is in state `START`, draw a `"white"`rectangle. Draw the next four states as rectangles with colors `"pale green"`, `"spring green"`, `"lime green"`, and `"dark green"`, respectively.

Suggestion: Draw these rules as a diagram in the style of Exercise 109, to help you as you develop your code. You don't need to hand this in, but having a picture available to you might help organize your thinking.

Reminder: This problem asks you to *design* the world program. That implies you must first determine what information should be in the world state, and design a data definition for it. Only then can you begin to design the helper functions that the world program needs in order to run.