CS 131 - Homework 4

Deadline:

5:00 pm on Friday, September 24

How to submit:

When you are done with the following problems:

- save your resulting Definitions window contents in a file with the suffix **`.rkt`**
- transfer/save that file to a directory on **nrs-labs (**preferably in a folder/directory named `131hw4`)
- use **ssh** to connect to **nrs-labs**
- `cd` to the folder/directory where you saved it (`cd 131hw4` for example)
- use the `ls` command to make sure your `.rkt` file is really there
- use **`~st10/131submit`** to submit it, with a homework number of **4**
- make sure that `~st10/131submit` shows that it submitted your homework `.rkt` file

Purpose:

Practice writing data definitions for structs, writing templates for structs, and programming with structs

Important notes:

- Each student should work **individually** on this homework.
- You are expected to follow the Design Recipe for all functions that you write.
  - Remember, you will receive **significant** credit for the signature, purpose, header, and examples/check-expects portions of your functions.
  - Typically you'll get at least half-credit for a correct signature, purpose, header, and examples/check-expects, even if your function body is not correct
  - (and, you'll **lose at least half-credit** if you omit these or do them poorly, even if your function body is correct).
- Be especially careful to include at least one specific example/check-expect for each "kind"/category of data, and (when appropriate) for boundaries between data. You can lose credit for not doing so.

The Problems:

Problem 0

Start up DrRacket, setting the language to **Beginning Student** and adding the HTDP/2e versions of the `image` and `universe` teachpacks by putting these lines at the beginning of your Definitions window:

```
(require 2htdp/universe)
```

```
(require 2htdp/image)
```

Put a blank line, and then type in a comment-line containing your name, followed by a comment-line containing CS 131 - Homework 4, followed by a comment-line with no other text in it --- that is,

```
; type in YOUR name
; CS 131 - Homework 4
;
```

Problem 1

Below what you typed in Problem 0 above, type the comment lines:

```
; Problem 1
;
```

Consider a setting that involves rhinos. Rhinos, in this setting, have a weight, a color, and an irritability index (which is in the interval [1, 10]).

1 part a

Write a data definition for a new struct type `rhino` (including a `define-struct` expression and the data definition comments, using the same style as demonstrated for `h-flier`).

1 part b

Now, within a comment, write the template for a function which expects a `rhino` parameter named `a-rhino`, using the same style as demonstrated for `h-flier`.

1 part c

Finally, within a comment, write the template for a function which produces a `rhino`, using the same style as demonstrated for `h-flier`.

Problem 2

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 2
;
```

Using the design recipe, design a function `safe-to-feed?` which expects a `rhino`, and produces `true` if it is safe to feed that rhino right now, and produces `false` if it is not. How do you know? If his/her irritability index is greater than or equal to 5, then it is NOT safe to feed that rhino right now...

Problem 3

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 3
;
```

Using the design recipe, design a function `add-weight` which expects a `rhino` and how much additional weight it has gained since its last weighing, and produces a new `rhino` whose weight now includes that additional weight.

Problem 4

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 4
;
```

Consider a ball, a lovely round orb that bounces up and down. We decide that, for our purposes, every ball has at least:

- a color
- a **diameter** in pixels
- the current x-location of its center
- the current y-location of its center
- its speed, which for a ball is how many pixels it goes up or down on each clock tick

(and you may add additional characteristics IF you wish).

If a ball's speed is positive, the ball is assumed to be going down, and if its speed is negative, it is assumed to be going up.

4 part a

Write a data definition for a new struct type `ball` (including a `define-struct` expression and the data definition comments, using the same style as demonstrated for `h-flier`).

4 part b

Now, within a comment, write the template for a function which expects a `ball` parameter named `a-ball`, using the same style as demonstrated for `h-flier`.

4 part c

Finally, within a comment, write the template for a function which produces a `ball`, using the same style as demonstrated for `h-flier`.

Problem 5

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 5
;
```

Using the design recipe, design a function `create-ball-image` that expects a `ball` instance, and produces an appropriate image depicting that ball, given its diameter and color.

(Please note! This function does not produce a `scene` -- it simply produces an `image`...)

Problem 6

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 6
;
```

During these next few problems, you will write some named constants and functions for a world consisting of a `ball` instance.

6 part a

Write named constant definitions for a `WIDTH`, `HEIGHT`, and `BACKDROP` for a scene that will eventually contain a ball instance.

6 part b

Now, using the design recipe, develop a function `draw-ball` that expects a `ball` and produces a scene depicting that ball (and making use of the `WIDTH`, `HEIGHT`, and `BACKDROP` you have just declared).

Problem 7

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 7
;
```

Using the design recipe, develop a function `ball-inside?` that expects a `ball` and produces whether or not the y-coordinate of its center is currently within the scene.

Problem 8

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 8
;
```

Consider how a bouncing ball might change based on user keystrokes. (Its diameter might grow or shrink; its color might change, given that an alternative color value in Racket is the result of the function `(make-color red-val green-val blue-val)`, where each of the arguments is in the interval [0, 255]; its speed might increase or decrease; its x-coordinate might be changed some amount; etc.)

8 part a

Decide on at least 3 keystrokes that will somehow change your bouncing ball, and, in a comment, write a data definition for an enumeration type for this set of ball-affecting keystrokes.

8 part b

Using the design recipe, develop a function `affect-ball` that expects a `ball` and an instance of your new enumeration type from Problem 8 part a, and produces the `ball` that should result from that keystroke.

Problem 9

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 9
;
```

Using the design recipe, develop a function `move-ball` that expects a `ball` and produces a new `ball` whose center in the scene is where the given ball's center should be after the next clock tick (based on its current speed). BUT note -- if the ball is outside of the scene when this is called, then the new ball's speed should be the negative of its current speed, and the new ball's position should be appropriately inside the scene.

Problem 10

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 10
;
```

Now bring this all together by using the design recipe to design a function `main` that expects an initial `ball` instance, and starts up `big-bang` with at least:

- that initial `ball` instance,
- `move-ball` as its `on-tick` function (and whatever your desired choice of clock-speed will be)
- `draw-ball` as its `on-draw` function
- `affect-ball` as its `on-key` function
- (and you may have additional `big-bang` clauses if you like)

This kind of `main` function is hard to write specific tests for -- so, no `check-expect`s are required for this `main` function.

Finally, call `main` at least twice, with at least two different initial `ball` instances.