CS 131 - Homework 5

Deadline:

5:00 pm on Friday, October 1

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 `131hw5`)
- use **ssh** to connect to **nrs-labs**
- `cd` to the folder/directory where you saved it (`cd 131hw5` 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 **5**
- make sure that `~st10/131submit` shows that it submitted your homework `.rkt` file

Purpose:

Practice writing data definitions for structs and lists and writing templates for functions involving structs and lists; practice designing functions involving lists (including lists of 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 5, followed by a comment-line with no other text in it --- that is,

```
; type in YOUR name
; CS 131 - Homework 5
;
```

Problem 1

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

```
; Problem 1
;
```

Copy the data definition for `number-list` from the posted in-class examples and paste it into your definitions window; also copy the template for a function `uses-num-list` that has a `number-list` parameter `num-list`.

Design a function `double-up` that expects a list of numbers, and produces a new list whose contents are each number from the original list multiplied by 2.

Problem 2

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

```
; Problem 2
;
```

(Adapted from Stephen Bloch's section of Adelphi's CS 160, Spring 2002)

2 part a

Develop a data definition for a `list-of-string`, and develop a template for a function `uses-strings` expecting a `list-of-string` parameter `string-list`.

2 part b

Design a function `count-string` that expects a string and a list of strings, and produces how many times that string occurs in the list.

Problem 3

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

```
; Problem 3
;
```

3 part a

Develop a data definition for a `list-of-image`, and develop a template for a function `uses-images` expecting a `list-of-image` parameter `image-list`.

3 part b

Design a function `scatter-images` that expects a list of images, and produces a scene with

those images centered in random locations in the scene. Make sure that each image's center is within the scene.

(You can and should write at least one `check-expect` for this function -- but in place of the other you should want to try to write, write an example call instead for that case.)

3 part c

Design a function `filter-images` that expects a maximum width, a maximum height, and a list of images, and produces a list of just those images from the original list whose width and height are strictly less than that given maximum width and maximum height.

Problem 4

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

```
; Problem 4
;
```

(Adapted from Cal-Poly San Luis Obispo TeachScheme!/ReachJava! Workshop, June 2008)

Consider a store that sells music CD's. For each CD, this store wants to keep track of its title, how many are in stock, and its category of music (such as "fuddy-duddy", "head-banging", "country", "western", etc.)

4 part a

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

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

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

Finally, write named constants to provide at least 3 examples of `cd` instances.

4 part b

Develop a data definition for a `list-of-cd`, and develop a template for a function `uses-cds` expecting a `list-of-cd` parameter `cd-list`.

Then, write named constants to provide at least 2 examples of `list-of-cd` instances.

4 part c

Design a function `total-stock` that expects a list of CDs and produces the total quantity of all of the CDs in that list.

4 part d

Design a function `category-stock` that expects a category of music and a list of CDs and produces a list of all of the CDs from that list that are in that given category and have at least one copy in stock.

4 part e

Design a function `category-titles` that expects a category of music and a list of CDs and produces a list of just the titles of all of the CDs from that list that are in that given category, regardless of how many copies are in stock.

Problem 5:

Skip a line, and write a comment noting that what follows is your work for

```
; Problem 5
```

Consider your `ball` struct from Homework 5. Copy its data definition, its two templates, and all of its associated functions into your Definitions window. (Note: you may also grab these from the posted example solution if you prefer.)

5 part a

Develop a data definition for a `list-of-ball`, and develop a template for a function `uses-ball` expecting a `list-of-ball` parameter `ball-list`.

5 part b

Develop a function `draw-ball-list` that expects a list of balls and produces a scene depicting those ball instances.

5 part c

Develop a function `move-ball-list` that expects a list of balls and produces the list of balls as they should be after the next clock tick.

(OPTIONAL: IF you wish, you may decide on some criteria, and `move-ball-list` can produce just the list of balls that meet that criteria as they should be after the next clock tick.)

5 part d

Consider your `affect-ball` function (or the posted example version). What would happen if you were to apply the action for each of those keystrokes to every ball in a list of balls?

Develop a function `affect-ball-list` that expects a list of balls and an instance of your keystroke enumeration type, and produces the list of balls as they should be as a result of that keystroke.

(OPTIONAL: IF you wish, you may define a new enumeration (or perhaps itemization) type, and have some keystrokes that cause balls to be added to or removed from the list of balls as a result of that keystroke.)

5 part e

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

- that initial `list-of-ball` instance,
- `move-ball-list` as its `on-tick` function (and whatever your desired choice of clock-speed will be)

- `draw-ball-list` as its `on-draw` function
- `affect-ball-list` 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 `list-of-ball` instances.