Lab 4

Goals and Instructions

Relevant sections of text: HtDP/2e Chapters 6, 7, 8; 9.0, 9.1; 10.0, 10.1, 10.2.

This week's lab will help you to practice:

- complex **itemizations** (e.g., itemizations of structures or other mixed types)
- simple functions on **lists** of structures.

Instructions. Please submit all answers via Sakai in a single, runnable file named lab4.rkt. Do all work in pairs. Include the names of both partners at the top of the file in a comment. You need only submit one lab per pair. *Make sure your submission runs without errors* (you can comment out sections that don't work for partial credit).

Add the following lines to the beginning of your file:

```
(require 2htdp/batch-io)
(require "google-maps-API.rkt")
```

The `batch-io` library allows you to use the function `write-file` to help you debug your program, as suggested in the lab.

The `google-maps-API.rkt` file must be downloaded with your `lab4.rkt` starter solution file from Sakai. *It must be in the same directory as your solutions file.* It contains a simple interface to the actual live Google Static Maps API (to be explained later)

It goes without saying, but you must use the Design Recipe for all functions. In particular,

- Don't forget to write a **signature**, **purpose statement**, and **unit tests** for every function.
- Write the tests **before** you write the function definition.
- **Break up functions into smaller functions if they become too complex.**
- Define every **Type** you use with a correct **Data Definition**.
    - Structure (Compound) Type definitions MUST include
        - Interpretation of fields
        - at least one example
        - the template
        - [Optional: write out the constructor, selector, and type predicate signatures if you are unsure of them]
    - Itemization (Mixture) Type definitions MUST include
        - the template
        - Note: you don't need examples, since the itemized choices already have examples
        - Note: you don't need an interpretation of fields, since an Itemization has no fields.

Mapping Things

With wearable and easily carried GPS devices like smart phones, mapping the locations of things has become a standard part of many of our daily lives. In this lab you will build some data structures to represent a few common mappable things, and an interface to all of them that can be saved to a file, or displayed graphically using Google's public static maps API (Application Programming Interface)—a web service.

It would not be a bad analogy to think of accessing services over the web (or in "the cloud") as ***calling a function*** (albeit remotely) that ***consumes*** certain structured data ***types*** and ***produces*** other structured data ***types***, according to a well defined ***signature***: exactly the kind of thing you should now be used to doing!

Realistic APIs (Application Programming Interfaces) to such data recognize three things:

1. There are many **different kinds of information** we might map. This implies **different data types** to represent this information computationally. *We will use **structures** to represent different kinds of information as different types.*
2. Our application needs to treat these data both **independently** (e.g. *"What is the Yelp! rating of this restaurant"*), and **collectively** (e.g. *"What things are nearby my current location?"*). *We will use an **itemization** of structures (mixture/union/interface) to write functions that work on all mappable info collectively; structures already work on independent map information.*
3. There is an **arbitrary number** of mappable things. *We'll use a **list** of mappable items to handle arbitrary amounts of information.*

Problem 1

Design a data type called `Restaurant` to represent a place that serves food. A `Restaurant` should contain a name, a phone number, a rating from 0 to 5 stars indicating the food quality, and a `GeoLoc`. `GeoLoc` should itself be a data type that you define. A `GeoLoc` should consist of a latitude [-90,90] and longitude [-180,180] in decimal degrees. Define other data types if appropriate. *[In a real system, this info might come from another API; in fact the sample data we provide is from Yelp!.*]

Problem 2

Design a data type called `Person` to represent your contacts. A `Person` should of course include a name, a phone number, and `GeoLoc`. You must re-use the `GeoLoc` data type definition from Problem 1, as well as add any new data definitions as required. *[In a real system, this data would come from two other APIs: your local contacts and some remote GPS service like Apple's FindMyFriends.]*

Problem 3

Design a data type called `GeoFence` to represent a geo-fenced area notification; this notification should be displayed if you are currently within the given radius. A `GeoFence` should include a `GeoLoc` indicating the center of the area, a radius in meters, and a short textual description. It goes without saying (this last time) reuse earlier definitions if useful, and define others if necessary. *[Again, this data might come from a different source we are not building for this lab.]*

Problem 4

Design a data type called `Mappable` that can either be a `Restaurant`, `Person`, `GeoFence`, or simply a `GeoLoc`.

Problem 5

Design a function `mappable->string` that accepts a `Mappable` and produces a string representation of the given `Mappable`.

A `Restaurant` should be displayed as follows:

```
"[3 stars] Santa Fe Mexican Grill [302-369-2500] <39.6837546,-75.7456899>"
```

A `Person` should be displayed as follows:

```
"Dr. Siegel [302-831-0083] <39.682531,-75.754329>"
```

A `GeoFence` should be displayed as follows:

```
"Velociraptor Paddock: (1000 Meters) <39.665648,-75.749031>"
```

A `GeoLoc` should be displayed as follows:

```
"Location <39.689071,-75.757207>"
```

For full credit you will need to develop *several* (at least 4) helper functions (***"one function, one purpose"***). Your helper functions should also prevent REPEATING code fragments (***"write it only once, use it everywhere"***)—such repetition is a maintenance nightmare, since every copy must be found/fixed/changed.

HINT: BSL will try to represent exact decimals as rational numbers:

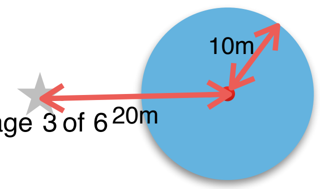
```
> (number->string 3.14)
"157/50"
```

In order to keep the decimals, convert them to inexact numbers first:

```
> (number->string (exact->inexact 3.14))
"3.14"
```

Problem 6

Design a function `distance` that accepts two `Mappable` arguments and produces the distance in meters between them. A `GeoFence` should be measured from the **outside** of the radius, not the center. So if I am 20 meters away from (the center of) a `GeoFence` of radius 10, then I am a distance of 10 meters away, because I am 10 meters away from the "fence" around the center point.



The star is 10m from the fence

Note that for full credit you will need to define helper functions as needed ***(I will stop repeating this now)***…

Because determining distance with latitude/longitude is, admittedly, difficult due to the Earth's curvature and no one remembering 9th grade conic section geometry, we have provided one helper function

```
;; gps-distance : Number Number Number Number —> Number
;; Parameters
;;  Number latA: latitude of point A
;;  Number lonA: longitude of point A
;;  Number latB: latitude of point B
;;  Number lonA: longitude of point B
;; Produces the approximate distance in meters between A and B
```

Problem 7

Design a function `nearby?` that consumes two `Mappable` arguments, and a distance in meters, and produces `#true` if the distance between the `Mappables` is strictly less than the given distance.

Define helper functions if needed for full credit on this problem. (Sorry. I'll stop now. Really.)

Problem 8

Design a data type called `ListOfMappable` that is a list of `Mappable`.

Problem 9

Design a function `select-nearby` that consumes a `ListOfMappable`, a `Mappable`, and a distance and produces a copy of the given `ListOfMappable` containing only the `Mappable` elements that fall strictly within the given distance.

Problem 10

Design a function `acceptable-restaurants` that consumes an `ListOfMappable` and a minimum star rating, and produces a `ListOfMappable` that includes only those restaurants on the list, if any, that meet or exceed the given rating criteria.

Problem 11

Design a function `mappables->string` that accepts a `ListOfMappable` and produces a `String` representation of the given `ListOfMappable` where each `Mappable` is displayed as in Problem 5, separated by a newline (`"\n"`) and in the order that they appear in the given `ListOfMappable`.

Note that in the interactions window and in your check-expect results, newlines will not be displayed. Instead you will see a literal \n.

To help debug string representations of this (or other large strings), you may wish to write your result to a file on your computer, which you can then view with your favorite text editor.

You can use the `write-file` function to write a string to a file. Calling `write-file` as follows will write the string "hello world" into a file called tmp.txt in the same directory as where your `lab4.rkt` file is stored.

```
(write-file "tmp.txt" "hello world")
```

When written to a file, all \n in a string will be converted to actual newlines. You can also write to the screen "Standard Output" by calling

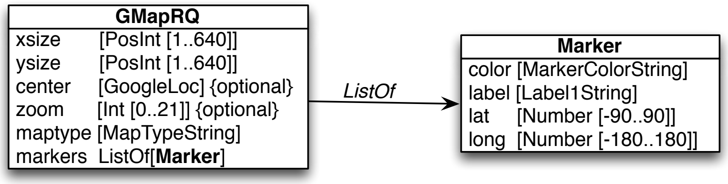
```
(write-file 'stdout "hello\n world\n")
```

Make sure you comment out all calls to `write-file` before you submit your lab. The TAs do not want a bunch of random files littering their drives.

Google Static Map API

Now, what would be better to show off your list of mappable things than an actual map? In order to draw the map, we'll call on Google's static map API *[Note: This API call to Google is LIVE and will require an active Internet connection.* ***DON'T ABUSE THE PRIVILEGE****. Google will ban your computer if you ask for thousands of maps without registering for developer access first.* ***You are restricted to 50 maps per minute, and 1000 maps per 24 hours.*** *We use only 1 map in this lab.]*

The Google API, just like the functions you write, uses some structures to indicate what kind of map tile to produce, and where you would like your map centered and where to place and style each marker. Sometimes programmers summarize large structure data definitions with a picture like the following (called a *UML [Unified Modeling Language] class diagram*). This picture shows you that a `GMapRQ` structure contains within it a list of `Marker` structures). **The other fields are either atomic types we already know, or simple enumerations or intervals on those types (note the use of both intervals and enumerations in Google's API).** A UML diagram of the Google Static Map Request API looks like this:


GMapRQ
xsize [PosInt [1..640]]
ysize [PosInt [1..640]]
center [GoogleLoc] {optional}
zoom [Int [0..21]] {optional}
maptype [MapTypeString]
markers ListOf[**Marker**]
ListOf
Marker
color [MarkerColorString]
label [Label1String]
lat [Number [-90..90]]
long [Number [-180..180]]

Note again the use of Structures, Intervals and Enumerations by Google. The fields `xsize`, `ysize` and `zoom` are ***intervals***, and the `maptype`, marker `color` and marker `label` are `String` ***enumerations*** (all data definitions details are documented in `lab4.rkt`). *For this lab, we will not be using the optional* `center` *and* `zoom` *parameters, so* `center` *can be the empty string* `""` *and* `zoom` *may be* `0`*.*

These structure definitions are already provided in `google-maps-api.rkt`; **you do not have to define them.** We have provided the full data definitions and templates in `lab4.rkt`.

Problem 12

Design a function `mappables->markers` that consumes a `ListOfMappables` and produces a `ListOfMarkers`. Restaurants should be colored blue and labeled "R"; People should be colored green and labeled "P"; geo-fenced areas should be colored red and labeled "F" (you will **not** need to draw the circle indicating the fenced area). Simple geolocs should be grey and labeled "G".

Problem 13

Develop the function `map-request` that consumes a `ListOfMappables` and produces a Google Static Map Request [`GMapRQ`]. We suggest a map width of about 400-500 pixels and the same for the height; these values should be defined as constants.

Problem 14

We have provided the following API for your use (again, remember this is a live call to Google's servers and requires an active Internet connection):

```
;; get-static-map-tile : GMapRQ —> Image
;; Parameters
;;   GMapRQ m: a Google static Map tile Request
;; Produces image of requested map tile containing all markers
```

Write two function calls to `get-static-map-tile`:

- One producing a map with all three types of markers (your choice).
- One producing a map of restaurants with more than 3 stars near to a location of your choice <lat,long>.