COMP 50 Lab: The Nearest Hospital

November 6–7, 2013

Over the next two lab periods, you will have a chance to build an interactive graphics application that finds the hospital closest to the mouse.

How to install the geographic names

The USGS names database has over 2 million names. To make the lab more manageable I've installed about 10,000 of them in a Racket package. Before you can use the names, you'll need to install this package.

1. Launch DrRacket
2. From the File menu choose Install Package
3. In the "Package Source" box, type

   `http://www.cs.tufts.edu/comp/50/packages/geonames.zip`

You should get a separate window that has a bunch of `raco setup` notifications. Probably the last one is

```
raco setup: --- post-installing collections ---
```

As soon as the Close button lights up (black text instead of gray text), press it.

Increasing the memory limit

Go the Racket menu, choose the "Limit Memory…" option, and set the memory limit to 512MB.

Testing the installation

Unfortunately it is not possible to install new teachpacks on the teachpack menu. Instead you have to put

```
(require geo)
```

into your source code.

If your installation has worked correctly, you should be able to do this in DrRacket's Interactions window:

```
> (require geo)
> (length geonames)
10150
> (third geonames)
(make-usgs "Bailey Creek" "Stream" "New Haven" "CT" 41.2737092 -72.6450939)
> (fifth geonames)
(make-usgs "Bates Brook" "Stream" "Fairfield" "CT" 41.4675943 -73.4756786)
```

Start a source file for your lab with the line

```
(require geo)
```

This line will give you access to `geonames`, which is a `(listof point-of-interest)`.

Data definitions

For the lab you will use a number of data definitions:

A. *Degrees* is a number measuring an angle in degrees.

B. A *feature-class* is one of the following strings: `"Airport"`, `"Arch"`, `"Area"`, `"Arroyo"`, `"Bar"`, `"Basin"`, `"Bay"`, `"Beach"`, `"Bench"`, `"Bend"`, `"Bridge"`, `"Building"`, `"Canal"`, `"Cape"`, `"Cemetery"`, `"Census"`, `"Channel"`, `"Church"`, `"Civil"`, `"Cliff"`, `"Crater"`, `"Crossing"`, `"Dam"`, `"Falls"`, `"Flat"`, `"Forest"`, `"Gap"`, `"Glacier"`, `"Gut"`, `"Harbor"`, `"Hospital"`, `"Island"`, `"Isthmus"`, `"Lake"`, `"Lava"`, `"Levee"`, `"Locale"`, `"Military"`, `"Mine"`, `"Oilfield"`, `"Park"`, `"Pillar"`, `"Plain"`, `"Populated Place"`, `"Post Office"`, `"Range"`, `"Rapids"`, `"Reserve"`, `"Reservoir"`, `"Ridge"`, `"School"`, `"Sea"`, `"Slope"`, `"Spring"`, `"Stream"`, `"Summit"`, `"Swamp"`, `"Tower"`, `"Trail"`, `"Tunnel"`, `"Unknown"`, `"Valley"`, `"Well"`, or `"Woods"`.

C. A *point-of-interest* is a structure:

```
(make-usgs name class county state latd lond)
```

where

- `name` is a string
- `class` is a *feature-class*
- `county` is a string representing a county name
- `state` is string containing the two-letter postal abbreviation for an American state
- `latd` is the point's latitude in degrees
- `lond` is the point's longitude in degrees

D. As on the homework, A `(2Dpoint X)` is a structure

```
(make-point x y value)
```

where `x` and `y` are numbers and `value` is an *X*.

E. A `(bbox X)`, also known as a *bounding box measured with X*, is a structure:

```
(make-bbox south north west east)
```

where `south`, `north`, `west`, and `east` are X's.

Functions defined for you

Standard functions on lists

At http://docs.racket-lang.org/htdp-langs/intermediate.html#%28part._htdp-intermediate._.Higher-.Order_.Functions%29 you will find a list of all the predefined abstract list functions. Here are some of the important ones, most of which we've discussed in class:

- `andmap` and `ormap`
- `filter`
- `map`
- `argmin` and `argmax`
- `foldl` and `foldr`

Functions specific to this lab

Here's a function that can tell you if a point of interest lies in a bounding box. The function returns a function!

```
; within-bbox : (bbox degrees) -> (point-of-interest -> boolean)
; to return a function that tells if a given point-of-interest is
;                                    within the given bounding box
(define (within-bbox bb)
  (local [(define (within? pt)
            (and (<= (bbox-west bb) (usgs-lond pt) (bbox-east bb))
                 (<= (bbox-south bb) (usgs-latd pt) (bbox-north bb))))]
         within?))

(define bb1 (make-bbox 42 43 -72 -70))
(check-expect ((within-bbox bb1)
               (make-usgs "Boston" "test pt" "Middlesex" "MA" 42.33 -71)) true)
(check-expect ((within-bbox bb1)
               (make-usgs "North" "test pt" "Middlesex" "MA" 43.33 -71)) false)
(check-expect ((within-bbox bb1)
               (make-usgs "STL" "test pt" "?" "MO" 38.6  -90.2)) false)
```

The next function wraps a point of interest inside a `2Dpoint`. It converts the GPS coordinate system to (x, y) coordinates suitable for use in making flat maps. There is some distortion; points at higher latitudes appear further apart than they actually are on the surface of the Earth. You can download both functions from http://www.cs.tufts.edu/comp/50/labs/hospitals.rkt.

```
; xy-converter : bbox number -> (point-of-interest -> (2Dpoint point-of-interest))
; to return a function that attaches (x, y) coordinates to the given point-of-interest,
; relative to the given bbox.  Height is the height of the bbox in pixels
(define (xy-converter bb height)
  (local
   [(define pixels-per-latitude-degree ; # of pixels representing 1 degree of latitude
      (/ height (- (bbox-north bb) (bbox-south bb))))

    (define pixels-per-longitude-degree ; # of pixels representing 1 degree of longitude
```

```
      (* (cosd (average (bbox-north bb) (bbox-south bb)))
         pixels-per-latitude-degree))

    ; x-in-pixels : degrees -> number
    ; convert the given longitude to an x distance from the west boundary
    (define (x-in-pixels x-degrees)
      (* pixels-per-longitude-degree (- x-degrees (bbox-west bb))))

    ; y-in-pixels : degrees -> number
    ; convert the given latitude to a y distance down from the north boundary
    (define (y-in-pixels y-degrees)
      (* pixels-per-latitude-degree (- (bbox-north bb) y-degrees)))

    ; convert : point-of-interest -> (2Dpoint point-of-interest)
    ; wrap `pt` in its (x, y) coordinates within `bb`
    (define (convert pt)
      (make-point (x-in-pixels (usgs-lond pt))
                  (y-in-pixels (usgs-latd pt))
                  pt))]
   convert))

; cosd : degrees -> number
; to return the cosine of the given number of degrees
(define (cosd degrees)
  (cos (* pi (/ degrees 180))))

(check-within (cosd 45) (/ (sqrt 2) 2) EPSILON)
(check-within (cosd 90) 0 EPSILON)

; average : number number -> number
; to return the average of the two given numbers
(define (average x y)
  (/ (+ x y) 2))

(check-expect (average 6 7) 6.5)


(define c1 (xy-converter (make-bbox 42 43 -72 -70) 100))

(define west (make-usgs "West" "test point" "?" "MA" 42.5 -72))
(define wnw  (make-usgs "West Northwest" "test point" "?" "MA" 42.75 -72 ))
(define nw   (make-usgs "Northwest" "test point" "?" "MA" 43 -72))
(define ne   (make-usgs "Northeast" "test point" "?" "MA" 43 -70))
(define sw   (make-usgs "Southwest" "test point" "?" "MA" 42 -72))
(define sse  (make-usgs "South Southeast" "test pt" "?" "MA" 42 -70.5))
(define mid  (make-usgs "Middle" "test pt" "?" "MA" 42.5 -71))

(define shrink (cosd 42.5))

(check-within (c1 west) (make-point 0 50 west) EPSILON)
(check-within (c1 wnw) (make-point 0 25 wnw) EPSILON)
(check-within (c1 nw) (make-point 0 0 nw) EPSILON)
(check-within (c1 sw) (make-point 0 100 sw) EPSILON)
(check-within (c1 ne) (make-point (* 200 shrink) 0 ne) EPSILON)
```

```
(check-within (c1 sse) (make-point (* 2 shrink 75) 100 sse) EPSILON)
(check-within (c1 mid) (make-point (* 2 shrink 50) 50  mid) EPSILON)


(define EPSILON 0.0001)
```

Functions and values to define for lab

All of these problems are to be solved **without defining any *recursive* functions**:

1. Define a function `all-h?` which is given a list of points of interest and tells if every point is classified as a hospital.
2. Define a function `any-h?` which is given a list of points of interest and tells if any point is classified as a hospital.
3. Define a function `hospitals` which is given a list of points of interest and produces a list of all the given points that are hospitals.
4. Define a function `easternmost` which is given a nonempty list of points of interest and returns the one with the easternmost location (largest longitude).
5. (*Optional*) Define a general-purpose function `how-many?` for answering list questions that begin with "how many." Do *not* use the `length` function; use `foldl` or `foldr`.
6. Define a function `hospital-fraction` that says what fraction of a given nonempty list of points of interest are hospitals. You may use `length` *once*.
7. Define a function `place-2Dpoints` which is given an image and a list of `(2Dpoint X)` and returns a copy of the original image in which each point is marked by a solid black circle 2.5 pixels in diameter.

   Teachpack knowledge: You will find `place-image` helpful here.
8. Define a value `northeast` which is a bounding box bounded on the south by 38 degrees, on the north by 48 degrees, on the west by -91 degrees, and on the east by -67 degrees.
9. Using function `within-bbox` and value `northeast`, the following code defines a function `northeast?` that tells if a given point of interest lies within the `northeast` bounding box:

   ```
   (define northeast? (within-bbox northeast))
   ```

   Using the `northeast?` function, define a value `northeast-points` that contains all the points of interest that lie within the `northeast` bounding box.
10. Define a value `northeast-hospitals` that contains all the *hospitals* that lie within the `northeast` bounding box.
11. Using the function `xy-converter`, define a function `as-northeast-point` that converts a given point of interest into a `(2Dpoint point-of-interest)` with `x` and `y` coordinates suitable for a map of the `northeast` bounding box. The map is planned to be 720 pixels high.
12. Define a function that is given (x, y) pixel coordinates and a *nonempty* list of `(2Dpoint point-of-interest)` and returns the point of interest that is *closest* to the given point. (If a particular 2D-point is located at (x_p, y_p), its distance from (x, y) is approximately the square root of $(x_p - x)^2 + (y_p - y)^2$.[1]

[1] Hint: the closest point is the one with the smallest square distance—you don't need to compute the square root.

13. Located at 40.232364 degrees North and 86.618958 degrees West (aka -86.618958 degrees longitude) is a lonely stretch of highway surrounded by corn and soybeans. Convert this location to a 2D-point and compute the name of the nearest hospital.

14. Create a map 1300 pixels wide and 720 pixels high containing a marker for every point of interest in the `northeast` bounding box.

    Teachpack knowledge: You will find `empty-scene` helpful here.

15. Create a map 1300 pixels wide and 720 pixels high containing a marker for every *hospital* in the `northeast` bounding box.

Submitting the lab

Ten minutes before the end of the lab, put the following text at the beginning of a DrRacket file. You may use an empty file or the source code you have been working on:

```
#|
What I did during this lab:
   (you fill in this part)

What I learned during this lab:
   (you fill in this part)

|#
```

Finally, *submit this file through the handin server* as `lab-hospitals`. **Submit it as lab, not as homework.** You will need to submit it using *two* usernames connected by a + sign, as in

```
Jane.Doe+Richard.Roe
```

You submit using Jane Doe's password.