# Homework: Structures and variants, part I

COMP 50

Fall 2013

This homework is due at 11:59PM on **Monday, September 23**.
Submit your solutions in a single file using the COMP 50 Handin button on DrRacket; the homework is the `structures-i` homework.

All the exercises should be done using the *Beginning Student Language*.

*Hint*: throughout the homework there are a number of uses for the functions `round`, `quotient`, and `remainder`.

## Finger Exercises

For this homework I am recommending the following finger exercises:

- In the First Edition *How to Design Programs*, Exercises 6.4.1, 6.4.3, 6.5.1, 6.6.1 (but instead of a symbol for color, please use a string), 6.6.3, 7.1.1, 7.1.3, 7.2.1, 7.2.2, 7.5.1, 7.5.3
- Develop data examples for the structure definitions in Exercise 6.4.1
- Write a function that converts a GPS position (as you define it below) to English

## Problems to submit

1. Write a *structure definition* and a *data definition* for representing clock time as a number of hours, minutes, and seconds elapsed since midnight. Unlike the example solution provided on the HtDP web site, your data definition must make clear exactly which values are and are not acceptable clock times.

   *Define a function* `move-big-hand` which adds one minute to a time structure.

   *Define a function* `time->digital` which consumes a time structure and produces an *image* showing how the time would read on a digital clock. Use the design recipe for a function that consumes compound data.

2. The Garmin eTrex handheld GPS renders distances in three ways:

   - Any distance up through 999 feet is rendered to the nearest whole foot, followed by the abbreviation "ft".
   - Any distance from 1000 feet up through 999 miles is rendered to the nearest *tenth* of a mile, using a decimal point and the abbreviation "mi".
   - Any distance of 1000 miles or more is rendered to the nearest mile, with no decimal point and the abbreviation "mi".

   *Define a function* `meters->english` which converts a distance in meters to a string showing the English units as on the Garmin GPS. Example outputs include `"37ft"`, `"850ft"`, `"0.2mi"`, `"10.0mi"`, and `"2268mi"`.

   Use the design recipe for conditional functions.

   **Domain knowledge**: 1 inch is 2.54 centimeters. 1 foot is 12 inches. 1 mile is 5280 feet.

   For test cases you can do Google searches such as "500 meters in feet" or "500 meters in miles".

3. *Define a structure* `gps-posn` and a *data definition* which represents GPS coordinates (latitude and longitude). Be sure that the data definition is sufficient so that anyone can determine whether an example structure is or is not a good GPS position.

   *Define a function* `checked-make-gps-posn` which behaves like `make-gps-posn` except that it guarantees that the latitude and longitude are good.

   *Define a function* that returns the distance in meters between two *nearby* GPS positions, assuming that near the positions you are interested in, the Earth's surface has been flattened. You may wish to consult the handout on the Earth's geometry.

   A *bearing* is a direction given relative to the compass. A bearing may be given as a string (N, S, E, W, NE, SE, NW, and SW) or as a number of degrees between 0 and 360. A bearing of zero degrees is North, ninety degrees is East, and so on. *Write a data description* for bearings.

   *Define a function* which given two nearby GPS positions, returns the bearing of the second point with respect to the first. This bearing is the direction you are facing if you stand on the first point and look at the second. Again, assume that the Earth's surface has been pressed flat near the positions you are interested in. *Potential trap:* if a trigonometric function returns a negative number of radians, in order to convert to a bearing, you will need to come up with a positive number.

   *Define a function* `gps-project` which consumes a position, bearing, and distance, and produces a new position obtained by starting at the first position, facing the given bearing, and traveling the given distance. This function should assume the Earth's surface has been flattened at the location of the initial point. This function (with a better model of the Earth) is built into my handheld GPS unit.

   Use both the design recipe for compound data and the design recipe for mixed data.

   **Domain Knowledge**: I've created a handout that shows the trigonometry needed to do the computations under the assumption that near the points of interest, the Earth's surface has been pressed flat. Here is some additional knowledge about the domain:

   - At any point on the Earth's surface, travel north or south along a line of longitude traverses one minute of arc for every nautical mile traveled.
   - Travel east and west along the Equator also traverses one minute of arc for every nautical mile traveled.
   - As you move north or south, the lines of longitude get closer together, and traversing a minute of arc in an East-West direction requires that you travel *less* than one nautical mile.
   - Using the approximate (flattened surface) model of the Earth, for computations within the Boston area you can expect to compute bearings that are accurate to within about 1/10 of a degree, and you can probably compute distances traveled accurately to within a few percent.

   The information about arc-minutes and nautical miles should help you write test cases.

   *Hints*: Using sines and cosines will put *inexact* numbers in your GPS-position structures. The `check-expect` form will not work with such structures. I recommend that you define a function `gps-equal-within?` which you can use as follows:

```
(check-expect (gps-equal-within? p1 p2 EPSILON) true)
```

## Karma problems

A. *Define a function* `time->analog` which converts a clock time into an image representing an analog clock. Your image must include an hours hand and a minutes hand; whatever other elements you wish to include are up to you.

B. *Use* `big-bang` to animate the passage of time on a clock. Run the animation at 360 times real time.

C. Your `gps-project` function is already required to accept the four "cardinal directions" (N, S, E, and W) as well as the "intercardinal" directions NE, SE, NW, and SW. For good karma, *extend your `gps-project` function* so that it also accepts the eight "half-winds," such as NNE, and the eight "quarter-winds", such as WbN (West by North). Your final function should work with all 32 named points on the Compass Rose.[1]

## How your work will be evaluated

Here's what we'll be looking for. First, the items of national-security importance:

- *Data descriptions*
  - We expect complete data descriptions including invariants (i.e., properties or relationships among parts of the data).
  - We expect data descriptions even for interval data, such as distance or time.
- *Helper functions*
  - A single function should do one *small* job. All three problems, to be designed well, need helper functions.
- *Examples and tests*
  - Both the time problem and the GPS problem are rife with possibilities for error, including nonsensical time displays and GPS projections that arrive at the wrong destination.
  - The GPS problem in particular is difficult to test. We expect two kinds of tests:
    * Tests based on *external* information, such as the direction of Boston when standing in Somerville. You can gather external information from Web resources or you can take a compass to the top of Tisch library and sight on prominent buildings.[2]
    * Tests based on *internal consistency*, such as if I go a mile north, then a mile south, I should end up close to where I started.

Other parts of the design recipe are also important:

- *Signatures, purpose statements, and headers*
  - The key property here, as always, is that your signature, purpose statement, and header should be precise enough to characterize *examples*.
- *Function templates*
  - Conditional structures should follow an *explicit* description or analysis of data.
  - Functions consuming structures should respect the design template for consuming compound data.

[1] A famous mariner once said that 32 directions ought to be enough for anyone.
[2] Be alert for the difference between magnetic north and true north.