CS 2500, Spring 2013
Problem Set 8

Due date: Tuesday, March 19 @ 11:59pm

Programming Language: Intermediate Student Language

You must work on and submit this homework with your assigned partner. You will receive no credit for ignoring your partner and working on the homework alone or not working on the homework at all.

Using Blackboard, submit a single Racket file containing all of the code and documentation for this assignment. Place your names and husky email addresses in a comment at the beginning of your file.

Name your file: hw8-lastname1-lastname2.rkt

You must follow the design recipe in your solutions: graders will look for data definitions, contracts, purpose statements, examples/tests, and properly organized function definitions. For the latter, you must follow templates. You do not need to include the templates with your homework, however, unless the question asks for it.

Problem 1:
Here is a data definition:

```
;; A family-tree-node is one of:
;; - empty
;; - (make-child father mother name date hair-color)
;; where father and mother are family-tree-nodes,
;; name and hair-color are symbols, and date is a number.
```

Use the above data definition to solve the following problems:

1. Develop the function count-older that consumes a family-tree-node and a year and returns the number of people in the tree that were born that year or earlier.
2. Develop the function search-tree-older that consumes a family-tree-node and a number and produces a list of all of the people in the tree born in that year or earlier.
3. Develop the function red-haired-ancestors? that determines whether a family-tree-node contains an ancestor with red hair on the father's side *and* an ancestor with red hair on the mother's side.
4. Develop the function update-father that consumes two family-tree-nodes and produces a family-tree-node that updates the father of the first tree to be the second tree.

Problem 2:
Review your code from the Frogger HW7 project. Rewrite the project according to the graders' mark ups. Be sure to fix any and all problems that your graders have discovered in your tests/functions.

We will get your submitted HW 7 files back by 3/11/12, but until then, you can work on Problems 1 and 3.

Problem 3:
Upgrade your Frogger program to full Frogger. This means that in addition to five rows of traffic, there are five rows of the river. In the river rows, the player must ride (collide with) a turtle or a plank at all times. All planks move right, and all turtles move left. When the player collides with a plank or turtle, they don't move relative to the entity. Turtles will always be above the water level. If the player hits a vehicle or is in the river but not on a plank or a turtle, the game ends with the player's loss. In addition, the player loses if he/she leaves the bounds of the screen. When any other entity (vehicle, plank or turtles) passes off the edge of the screen, a new entity of the same type is created on the opposite edge, so that there are always the same number of vehicles, planks, and turtles and they are all evenly spaced. Planks and Turtles will always have the same length and speed.

The player can move his character in four ways: up, down, left and right. When the player moves, the frog turns to match the direction that he just moved in. Develop your own data definitions and structures for Frogs, Vehicles, Planks, and Turtles (and any auxiliary ones you might need, like Worlds).

Optional: You may use images from frogger-library.rkt in your code. Be sure to follow the instructions in frogger-library.rkt if you wish to include the images. Thanks to Martha Hamlin for creating the images.

You may implement other features, if you like (e.g., multiple lives, planks and turtles of variable speed and length, or appearing and disappearing turtles). However, extra features won't save you from points taken off if your code has bugs or isn't well written. You will not receive 100% credit simply for having code that works. For full credit, your code must work and be well written. So you should put your effort into writing clean, readable, bug-free code.

Of course, the use of helper functions is suggested, and testing is not only helpful, but required.