Honors Homework 1

Discrete Structures, CS 173, Spring 2019

Due Sunday February 24th

This term, the first three assignments will use a dialect of Scheme called Racket. The goal of this homework is to get started with the basics: install Racket, get used to simple programming constructs.

This is a solo assignment. Do not share code or detailed designs with other students.

1 Installation, finding the manuals

The main Racket page is at

https://racket-lang.org/

Download and install racket on your computer.

Back on the main Racket page, select Docs at the top. Locate several useful things

- "Getting Started" explains how to power up the IDE
- "Quick: An Introduction to Racket with Pictures" the tutorial we'll be using
- "Racket Guide" language manual, useful later
- "Racket Cheat Sheet" list of functions, some are mysterious, some will make sense soon, some you can probably guess from the name

2 Getting started

Power up the IDE. Work through sections 1-7 of the Quick tutorial. Try their examples. Try simple modifications of their examples.

Don't worry if you don't understand absolutely everything. In particular, some of section 5 might be a bit hard to fully understand. Focus on copy-coding from their examples and getting used to the basic syntax.

On the upper menu, look for FILE and then "save definitions." This will let you save and load files of code ("definitions") that you've typed into the upper section of the IDE.

3 More basic syntax

For the rest of this assignment, you'll need a few arithmetic functions. Look in the "primitives" section on the Racket Cheat Sheet to find basic numerical functions. These are similar to the ones in other programming languages, except for having the parentheses outside the whole function call. E.g.

```
(define x 5)
(>= (+ x 3) 10)
```

should return #f (false). Try playing around with some of the numerical functions.

Also go to the Racket Guide and look at the section on "conditionals." This has examples of using the if statement, as well as logical connectives like AND. The when statement (see example below) is similar to if, except that it takes only two inputs: the test and what to do if the test succeeds.

4 Recursive functions

All dialects of lisp and scheme use recursive functions more often than languages such as C++ and Java. Here's a simple recursive implementation of the factorial function in Racket:

```
(define (factorial n)
   (when (< n 0) (error ‘‘Factorial can’t handle negative inputs’’))
   (if (= n 0)       ;; test
       1             ;; if
       (* n (factorial (- n 1)))))    ;; else
```

Notice that a semicolon makes the rest of the line into a comment.

```
(define (gcd a b)
(let ((r (remainder a b)))
  (if (= r 0)    ;; test
      b          ;; if
      (gcd b r))))  ;; else
```

In a similar style, write a recursive remainder function. Let's call it rem, so that the name won't clash with the built-in function. That is, the call

(rem x y)

should return the remainder of x divided by y. Assume without checking that both inputs are integers, but check that y is positive. Find the remainder by subtracting y from x, or adding y to x, until x is in the correct range for the remainder.

Make sure that your outputs follow the remainder definition from the textbook, even if the first input (x) is negative.

5 Recursive pictures

The quick tutorial shows how to make little graphical objects and glue them into sequences. Build a recursive function that takes an object O and a positive integer n as input, and produces a horizontal sequence of n copies of O. For reasons that will become obvious later, make this insert a small amount of space between the objects.

Your function must be recursive. Do not (for example) use the methods from section 8 of the Quick tutorial. And definitely don't go look up Racket support for things like arrays. You want something short that calls itself. Think about how to represent a sequence of n objects as one object plus a sequence of (n-1) objects.

Notice that the function should take an object as input, not (for example) the name of a function that creates objects (e.g. circle).

Write a similar function that makes a vertical sequence.

Use those two functions to write a function that returns a 2D grid of the input object. Its inputs should be the dimensions of the grid and the object.

6 Alternating objects

Finally, write a recursive function that makes an alternating sequence of two objects. This function should take a postive integer and two objects as input.

It should return a horizontal sequence that contains alternating copies of the two objects. Your function must work even when the input number is odd.

For full credit, your code should not test whether the input number is odd or even. It should be sufficient to test whether the input number has reached one.

7 Checkerboard

Using the methods from the previous two sections (and perhaps some of your previous functions) write a function that produces a checkerboard of two objects. The inputs should be the dimensions of the checkerboard (which might not be square) and the two objects.

8 Deliverables

You will need to submit a single racket file (with file type rkt) containing the following code:

- Your implementation of remainder
- Your recursive horizontal and vertical sequence functions, also your function for making a 2D grid.
- Your function for the horizontal alternating sequence of objects.
- Your function for making a checkerboard.

The functions should all be short, so the whole file should not be very long.

Your functions should use good style, e.g. blank line between functions, reasonable indenting, comment explaining (at least briefly) what each function does. It should be easy for me to load your file into racket and identify which of your functions implements each of the above.

Error checking is encouraged, especially because it will help you debug your code. But it doesn't need to be comprehensive. For example, don't worry that I might give negative dimensions to your checkerboard function.

To submit, find CS 196 (section 73) on moodle. If you aren't already in that section, hunt it down on the moodle class lists and self-enroll into it using the password Minerva. There will be an assignment activity on the moodle site, where you can upload your file of code.