CS 270 Lab 2 (Functional Programming in Racket and Map/Reduce)

Week 2 – Oct. 2 – Oct. 6, 2017.

Name 1:_____

Drexel Username 1:_____

Name 2:_____

Drexel Username 2:_____

Name 3:_____

Drexel Username 3:_____

Q1 (25%)_____ Q2 (25%) _____ Q3 (25%)_____ Q4(25%)_____.

Instructions: For this exercise you are encouraged to work in groups of two or three so that you can discuss the problems, help each other when you get stuck and check your partners work. There are four problems that ask you to implement recursive algorithms in the Racket programming language using the DrRacket IDE. The first two problems are to implement the higher-order functions map and reduce and the third and fourth problems utilize map and foldr (the racket version of reduce) to solve problems. The function map applies a function to the elements of a list, creating a new list consisting of the function applications. The function reduce combines elements of a list using a specified function. The two functions together provide a powerful programming paradigm and serve as the basis of Google's MapReduce language for processing and generating large data sets.

The last problem asks you to construct all tilings of a 2xn rectangle using 1x2 dominoes. Make sure you sufficiently test your programs that you are confident that they are correctly implemented, and when in doubt try examples of a library function before using it. This question is challenging.

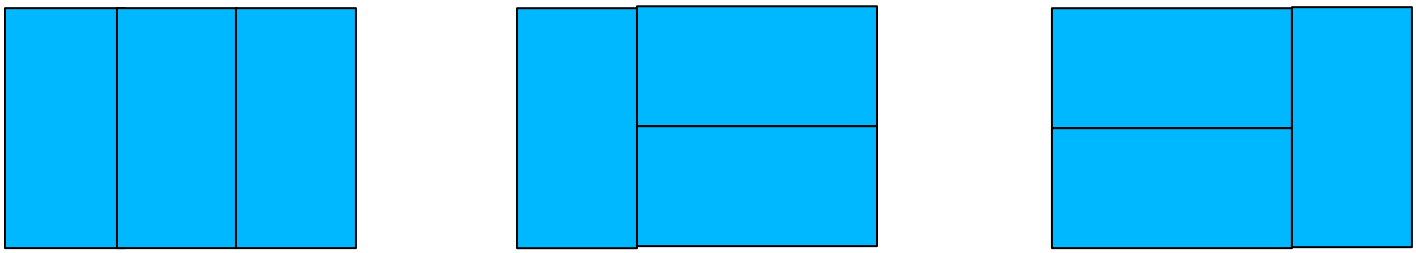
1. Implement the function (map f L) which takes two arguments: 1) f a function of a single variable and 2) L a list of elements in the domain of f. If L = (x1 x2 … xn), the output of the function is the list ((f x1) (f x2) … (f xn)). E.G. assuming the function sqr, defined by (define (sqr x) (* x x)), the result of (map sqr '(1 2 3 4)) is '(1 4 9 16). First try this example using the Racket function map and then see that you get the same result using your function. Note that Racket's map function is more general in that it allows functions of more than one variable when additional lists are provided.

2. Implement the function (reduce f init L) which takes three arguments: 1) f a function of two variables and 2) L a list of elements in the domain of f, and 3) the value to be returned when L = '(). Reduce is the same is the Racket function foldr. E.G. (reduce + 0 '(1 2 3 4)) is (+ 1 (reduce + 0

‘(2 3 4)) = (+ 1 (+ 2 (+ 3 (+ 4 0)))). First try the Racket function foldr, i.e. (foldr + 0 ‘(1 2 3 4)) and then see that your implementation obtains the same result. Try the other examples from the functional programming in Racket lecture.

3. Use map and reduce to solve the following problem: count the number of negative numbers in a list.

4. Given a set of 1 × 2 dominoes, write a Racket function to generate all of the different ways to to tile a 2 × n rectangle placing a single domino vertically (2 × 1) or stacking two dominoes horizontally? For example, for n=3, there are 3 ways.



Think recursively: What are the base cases and how many do you need? How do you solve the 2 × n using solutions to smaller problems?

- Represent tilings with “V” for one vertical tile and “HH” for two stacked horizontal tiles. In the above 2 × 3 example, the three tilings are represented by the strings “VVV”, “VHH”, and “HHV”. Note that all three solutions have length 3.
- Your function, (GenDominoes n) should return a list of strings, one for each possible tiling. In the above example, (GenDominoes 3) should return the list ‘(“VVV” “VHH” “HHV”)
- Your function should recursively generate all solutions of size n-1 and concatenate a “V” in front of each of them, and recursively generate all solutions of size n-2 and concatenate a “HH” in front of them. Use map to

perform the concatenation and use append to combine the solutions. What is the base case of your recursion?

- You should use Racket's built in string concatenation function: string-append. See the Racket documentation for details on string-append. Make sure you try it out before you write a function that uses it.