CS 270 - Lab 9

M. Boady, B. Char, J. Johnson, G. Long, S. Earth

1 Introduction

You may work in teams of **one** or **two** students. Submit one copy for the entire group.

Write your answers on this lab sheet. Only what is written on this lab sheet will be graded.

This lab is due at the end of the class period. You may not continue to work on it once class has ended.

This lab contains 4 questions.

Grading

- 25 points - Putting everyones names on this page
- 20 points - Earned for each correct question (Answer is fully correct)
- 5 points - Earned for **partial credit** any question

No additional point amounts can be earned. You cannot earn 7 points on a question for example.

The maximum score for a lab is 100. If you get everything correct, that adds up to 105 points but will be reduced to 100.

A question will be marked correct as long as it covers all requirements of the question. It does not need to be perfect, but must be fully correct. A single typo or very minor issue where the intention is clear and all requirements are met would still earn full points.

We want you to complete questions fully, not try to earn partial credit on multiple questions. You may ask your Professor/Course assistant questions during lab.

Labs must be done in the presence of an instructor and/or course assistant or credit will not be given.

Partners should alternate each class day which person is physically typing and submitting the lab.

Do not split up the problems or you risk not finishing on time due to the cumulative nature of the questions.

Enter the name of the student in the group

Member 1 (submitter): Elan Rubin

Member 2: Josh Koo

Question 1 :

Write a **recursive** Racket function that applies an arbitrary function f to the value n until the result is less that b.

Define the function (star f n b) to return the number of times the function *f* needs to be repeatedly applied until the input values is less that *b*.

```
;input-contract:afunctionfthattakesanintegeranddecreasesitsvalue,
;     n is a non-negative integer, b is the target to get below
;output-contract: an integer, the amount of times the function needs
;     to be applied repeatedly to get a value less than b
(define (star f n b)
        ; write your code in the space below the examples
)
```

Examples:

```
(star log 10 1) ;Returns 2
(star log 100 1) ; Returns 3
(star (lambda (x) (quotient x 3)) 242 1) ; Returns 5
(star sqrt 10 2) ;Returns 2
(star sqrt 500 2) ;Returns 4
(star (lambda (x) (- x 2)) 1000 2) ; Returns 500
```

```
#lang racket

(define (star f n b)
  (if (< n b)
      0
      ;
      (+ 1 (star f (f n) b))
    )
)
```

Question 2 :

The function (map f L) is built into Racket. It takes two arguments:

1. f a function of a single variable
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)). Map creates a new list by applying the function to each value in the list.

If we want to square numbers we know that is ($*$ x x). The result of (**map** (lambda (x) ($*$ x x)) ‘(1 2 3 4)) is ‘(1 4 9 16). It makes a list (1^2 2^2 3^2 4^2).

(a) Using lambda and map come up with a single line command to change the list ’(1 2 3 4) into the list ’(3 6 9 12)

```
(map (lambda (x) (* x 3)) '(1 2 3 4))
```

(b) Using lambda and map come up with a single line command to change the list ’(1 2 3 4) into the list ’(-1 -2 -3 -4)

```
(map (lambda (x) (- 0 x)) '(1 2 3 4))
```

(c) Using lambda and map come up with a single line command to change the list ’(1 2 3 4) into the list ’(9 8 7 6)

```
(map (lambda (x) (- 10 x)) '(1 2 3 4))
```

Question 3 :

Racket has two functions to combine the elements of a list. Both have the same three inputs.

1. f a function of two variables
2. *init* the value to be returned when L = ‘()
3. L a list of elements in the domain of f

Calling (foldr $-$ 0 ’(1 2 3 4)) computes -2

This function computes ($-$ 1 ($-$ 2 ($-$ 3 ($-$ 4 0)))).

Calling (foldl $-$ 0 ’(1 2 3 4)) computes 2

This function computes ($-$ 4 ($-$ 3 ($-$ 2 ($-$ 1 0)))).

Decide which direction to fold (foldr or foldr) and a single arithmetic operation starting at a base of either 0 or 1, such that the folding the list ’(1 2 3 4) would result in the target value. Write down the racket expression which accomplishes this.

(a) 24

```
(foldr * 1 '(1 2 3 4))
;or
(foldl * 1 '(1 2 3 4))
```

(b) $\frac{3}{8}$

```
(foldr / 1 '(1 2 3 4))
```

(c) $2\frac{2}{3}$

```
(foldl / 1 '(1 2 3 4))
```

Question 4 :

(a) Write a function (define (neg? x) ...) that returns 1 if the number is negative and 0 otherwise.

```
(define (neg? x)
 (if (< x 0)
 1
 0
 ))
```

(b) Write a racket expression that uses map to apply your neg? function to the list (1 −3 −4 5 9).

```
(map neg? '(1 -3 -4 5 9))
```

we initially used a lambda, but found it wasn't needed

(c) Write a single line expression to use map, foldr, and lambda to count the number of negative numbers in a list named L. Do not explicitly call the neg? function you wrote, instead use lambdas to do everything in a single line command.

```
(foldr + 0 (map (lambda (x) (if (< x 0) 1 0)) '(1 -3 -4 5 9)))
```