CS 2500 Exam 2 HONORS SUPPLEMENT – Fall 2014

Name: _____

Husky email id: _____

- This supplement to Exam 2 is intended for students enrolled in the Honors section of 2500.
- See the instructions on the regular exam, but keep in mind that specific instructions on any given problem override the general instructions on the regular exam. Also, you may use `lambda` or `local` as needed.

Problem	Points	/out of
1		/ 13
2		/ 12
3		/ 17
Total		/ 42

Good luck!

Problem 1 Design the function `partition`, which consumes a list and a test predicate on elements of the list, and partitions the list into two parts: elements that satisfy the predicate and those that do not. Your function should return a `split` structure comprised of two lists: `good`, which contains all elements of the input list that satisfy the test, and `bad`, which contains those that do not. Here is the structure type definition for `split`.

13 POINTS

```
(define-struct split (good bad))
```

Give `partition` its most general signature. Do not use recursion; instead use a loop function. Your solution must not traverse the input list more than once. The relative order of elements in the output lists should be the same as in the input list.

[Here is some more space for the previous problem.]

Problem 2 In the spirit of international collaboration, and thanks to the recent success of Philae the plucky comet adventurer, NASA is designing another comet probe. Since the ESA's probe was so successful, they plan to work with the ESA's software as much as possible, while reusing as much of their own code as possible, too. Unfortunately, NASA is still stuck using Imperial units of measure—their software assumes the spacecraft will operate in temperatures measured in Fahrenheit, whereas the ESA's software works with Celsius temperatures. 12 POINTS

Help NASA back into space by designing a function `celsius-fun->fahren-fun`, which given a function from Celsius to Celsius produces a function from Fahrenheit to Fahrenheit. Here are the relevant data definitions and the signature for the function:

```
;; Celsius = Number
;; interp: a temperature measurement in degrees Celsius

;; Fahren = Number
;; interp: a temperature measurement in degrees Fahrenheit

;; celsius-fun->fahren-fun :
;;    [Celsius -> Celsius] -> [Fahren -> Fahren]
```

You may use the following functions, if you wish.

```
;; c->f : Celsius -> Fahren
;; Convert Celsius temperature to Fahrenheit
(define (c->f c) (+ (* c (/ 9 5)) 32))

;; f->c : Fahren -> Celsius
;; Convert Fahrenheit temperature to Celsius
(define (f->c f) (* (- f 32) (/ 5 9)))

(check-expect (c->f 0) 32)
(check-expect (f->c 32) 0)
(check-expect (c->f 100) 212)
(check-expect (f->c 212) 100)
```

[Here is some more space for the previous problem.]

Problem 3 Consider the following data definition: 17 POINTS

```
;; A Toll is a Number

(define-struct node (toll left right))
;; A TollBT is one of:
;; - Toll
;; - (make-node Toll BT BT)
;; interp: The nodes and leaves of a TollBT represent exits;
;;   the toll info at each node/leaf gives the toll paid at
;;   that exit.

;; A SumTollBT is one of:
;; - Toll
;; - (make-node Toll BT BT)
;; interp: The nodes and leaves of a SumTollBT represent exits;
;;   the toll at each node/leaf gives the sum of the tolls
;;   paid to get to that exit from the root of the tree.
```

Design a function `tollBT->sumtollBT` that takes a `TollBT` and produces the corresponding `SumTollBT`. Here is an example to illustrate:

```
(tollBT->sumtollBT
    (make-node 4 (make-node 3 2 1) (make-node 1 3 2)))
--> (make-node 4 (make-node 7 9 8) (make-node 5 8 7)))
```

Hint: use an accumulator.

[Here is some more space for the previous problem.]