CS2500 Exam 2 — Fall 2014

Name: _____

Husky email id: _____

Section (Ahmed/Lerner/Razzaq/Shivers): _____

- Write down the answers in the space provided.
- You may use the usual primitives and expression forms, including those suggested in hints; for everything else, define it.
- To save time writing, you may write
  `(sqr 3)` $\rightarrow$ `9`
  instead of
  `(check-expect (sqr 3) 9)`
  You may also write the Greek letter λ instead of `lambda`, if you wish.
- Some basic test taking advice: Before you start answering any problems, read *every* problem, so your brain can be thinking about the harder problems in background while you knock off the easy ones.

Good luck!

Problem	Points	/out of
1		/ 16
2		/ 15
3		/ 16
4		/ 30
5		/ 54
Extra		/ 15
Total		/ 146
Base	**131**	

Problem 1 Give the signatures for these two functions. 16 POINTS

```
(define (f op elements)
  (map (lambda (elt) (+ 10 (op elt)))
       elements))
```

```
(define (g op1 op2)
  (lambda (x) (op1 (op2 (op1 x)))))
```

Problem 2 Consider the following data definition: 15 POINTS

```
;;; A NumTree (binary tree with numbers at the leaves) is one of:
;;; - Number
;;; - (make-node NumTree NumTree)
(define-struct node (left right))
```

Design a function, `count-winners`, that consumes a tree and a test predicate, and returns the number of leaves that satisfy the predicate (that is, the number of leaves that cause the test to return true.) For example, we could use `count-winners` to determine that `tree1` below has 3 positive leaves, or 2 odd leaves, *etc.* (You may refer to `tree1` in your tests/examples.)

```
(define tree1 (make-node 3
                         (make-node (make-node 8 0)
                                    7)))
```

[Here is a blank page if you need more space to write answers.]

Problem 3 Continuing on with `NumTrees` from problem 2, we can define a data definition that gives a path through a tree, from the root to some subtree, as a list of left/right directions:

16 POINTS

```
;;; Dir = 'left | 'right
;;;
;;; Directions = [List-of Dir]
;;;
;;; MaybeDirections is one of:
;;; - 'fail
;;; - Directions
```

So, for example, a `Directions` of `'(left left right)` means: "Start at the root of the tree, go to its *left* child, then go to that node's *left* child, then go to that node's *right* child."

Design a function, `tree-nav`, that navigates a tree: it consumes a `NumTree` and a `Directions`, and produces the subtree you obtain by starting at the root of the given tree and following the directions.

For example, following the directions `'(left left right)` when given the tree

```
(define tree1 (make-node (make-node (make-node 8 0)
                                    (make-node 1 3))
                         (make-node (make-node 2 7)
                                    (make-node 9 4))))
```

would produce the subtree 0. (Note that *this* set of directions got you all the way to a leaf, but other directions, such as `'(left left)` might only get you to an internal `node` structure, which is perfectly fine.)

You may assume that the directions are valid directions for the given tree (that is, you don't need to explicitly handle "bad directions").

If you wish, you may refer to `tree1` in your examples.

[Here is a blank page if you need more space to write answers.]

Problem 4 The *composition* $f \circ g$ of two functions f and g is defined by the equation 30 POINTS

$$(f \circ g)(x) = f(g(x)).$$

- **(11 pts)** Design the function `compose`, which takes two functions, f and g, and returns their composition $f \circ g$. (Caution: it's easy to get parts of the signature flipped around—better double-check this carefully.)

  (Note: You can't use `check-expect` to compare two functions. But you can apply two number-producing functions to the same input, and compare the result numbers with `check-expect`...)

- **(19 pts)** Design the function `list-compose`, that takes one argument, a list of functions, and composes them together. For example,

  ```
  (list-compose (list cos sqr add1))
  ```

  produces the function $h(x) = \cos((x+1)^2)$.

  Your definition of `list-compose` may not explicitly recur, but you may use a loop function.

  You may assume that all functions in the list consume and produce the same kind of values. That is, `list-compose` may not be used to compose `even?` with `sqr`. (This will keep your signature manageable.)

[Here is a blank page if you need more space to write answers.]

Problem 5 You are working for a national grocery chain, designing software for a smartphone app that will help shoppers keep track of their shopping lists as they shop for groceries. You come up with the following data definition:

54 POINTS

```
;;; Kind = 'produce | 'meat | 'dairy | 'bakery
;;; An Item is a (make-item Kind String Number)
;;; ShoppingList = [List-of Item]

(define item (kind name price))
```

For example:

```
(make-item 'produce "Broccoli"    3.27) ; Broccoli is $3.27/bunch this week.
(make-item 'dairy   "Gallon milk" 3.75) ; A gallon of milk is $3.75.
```

For all parts of this problem, do not use explicit recursion. Instead, use a loop function—but do not use `apply`.

- **(15 pts)** Design a function, `total-price`, that consumes a `ShoppingList` and produces the total price of all the items in the list.
- **(13 pts)** Design a function, `expensive-items`, that consumes a price (a number) and a `ShoppingList`, and produces a list of all the items in the shopping list that exceed this price.
- **(13 pts)** Your grocery store has an early-bird special, that allows shoppers who purchase groceries between 4:00 am and 5:00 am to enjoy a discount of 7% on all items.

  Design a function, `discount`, that consumes a `ShoppingList` whose items have regular prices, and produces that shopping list, but with all prices reduced by 7%. (For example, if some item in the input list has a price of $10, it will have a price of $9.30 in the output list.)
- **(13 pts)** One of your checkout clerks is a committed vegetarian and refuses to deal with shopping carts that have any meat. (There's really nothing to be done—his mother is the CEO.) You've been asked for a function, `vegetarian?`, that determines if a `ShoppingList` is entirely free of meat items. Design this function.

[Here is a blank page if you need more space to write answers.]

[Here is *another* blank page if you need more space to write answers.]

Problem 6 (Extra credit) 15 POINTS

Leafy trees! Consider the following data definition:

```
;;; A Tree is one of
;;; - 'leaf
;;; - (make-node Tree Tree)
(define-struct node (left right))
```

Design a function, `leafy-trees`, that consumes a non-negative integer and produces the list of all leafy trees having the given number of leaves. For example, the list of all leafy trees having exactly three leaves is

```
(list (make-node 'leaf (make-node 'leaf 'leaf))
      (make-node (make-node 'leaf 'leaf) 'leaf)))
```