CS 2500 Exam 1 HONORS SUPPLEMENT – Fall 2012

Name: _____

Student Id (last 4 digits): _____

Instructor's Name: _____

- This supplement to Exam 1 is intended for students enrolled in the Honors section of 2500.
- See the instructions on the regular exam.

Problem	Points	/out of
1		/ 4
2		/ 6
3		/ 12
4		/ 14
5		/ 8
Total		/ 44

Good luck!

Problem 1 Recall Problem 4 from the regular portion of the exam. Your next task as consultant is to design a program, `donor-names`, that consumes a list of donations and produces a list of all the donors' names. 4 POINTS

Consider the following list of donations:

```
(define lod2012 (list 40
                      (make-donation "Vona" 2500)
                      (make-donation "Shivers" 100)
                      (make-donation "Shivers" 600)
                      31
                      (make-donation "Ahmed" 2000)
                      (make-donation "Vona" 2500)))
```

Given `lod2012` as input, `donor-names` should produce:

```
(list "Vona" "Shivers" "Shivers" "Ahmed" "Vona")
```

You may use the following data definition for a list of strings:

```
;; A LOS is one of:
;; - empty
;; - (cons String LOS)
```

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

Problem 2 The campaign would like to contact past donors to ask them to donate again. But there's a problem: the list of names produced by `donor-names` contains duplicates. The campaign doesn't want to send duplicate messages to the same donor—annoying one's supporters is not good strategy! 6 POINTS

Your next task is to design the `remove-duplicates` program which takes a list of strings and removes duplicates from the list.

For example:

```
(remove-duplicates
  (list "Vona" "Shivers" "Shivers" "Ahmed" "Vona")) --->
(list "Vona" "Shivers" "Ahmed")
```

You may design helper functions as needed, but they should be designed according to the recipe.

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

Problem 3 Recall Problem 5 from the regular portion of the exam where we were concerned about unscrupulous, wealthy donors getting around the laws on campaign donations by donating multiple times. The campaign you're working for would like to keep track of the total amounts donated by each named donor so that they can easily spot such violations. Your next task, therefore, is to design a program, `coalesce`, that consumes a list of donations and produces a list of donations in which multiple donations by the same named donor have been coalesced into a single donation of the total amount donated by that individual (and all anonymous donations in the input list are ignored).

12 POINTS

For example, given `lod2012` (from Problem 1) as input, we'd want `coalesce` to produce:

```
(list (make-donation "Vona" 5000)
      (make-donation "Shivers" 700)
      (make-donation "Ahmed" 2000))
```

You may design helper functions as needed, but they should be designed according to the recipe. Feel free to use any functions you've previously developed (on the honors supplement or the regular portion of the exam).

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

Problem 4 In computer science, programmers often work with binary trees. Binary trees are made up of *nodes* which have exactly two children (usually labeled `left` and `right`) that are themselves binary trees, and *leaves* (which have no children). 14 POINTS

For this problem, you will work with weighted binary trees, which are binary trees whose leaves carry weights. Here is a data definition for a weighted binary tree:

```
(define-struct node (left right))

;; A WBT (weighted binary tree) is one of:
;; - Weight               ; leaf, carrying a weight
;; - (make-node WBT WBT)  ; node with two children

;; A Weight is a Number
```

Design a program that determines if a weighted binary tree is balanced (i.e., there are no nodes in the tree whose `left` and `right` children carry unequal weights). Here are some examples of balanced weighted binary trees to help you out:

```
13
(make-node 13 13)
(make-node 13 (make-node 6.5 6.5))
```

And here are some examples of unbalanced weighted binary trees:

```
(make-node 13 6.5)
(make-node 13 (make-node 3 10))
(make-node (make-node 3 10) (make-node 3 10))
```

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

Problem 5 Using the data definition of weighted binary trees from the previous problem, design a program that consumes a weighted binary tree and produces a weighted binary tree that is the mirror image of the input. For example: 8 POINTS

```
13 ---> 13

(make-node 10 3) ---> (make-node 3 10)

(make-node (make-node 10 3) (make-node 6 8)) --->
(make-node (make-node 8 6) (make-node 3 10))
```