CS 2500 Exam 2 HONORS SUPPLEMENT – Fall 2012

Name: _____

Student Id (last 4 digits): _____

- This supplement to Exam 2 is intended for students enrolled in the Honors section of 2500.
- See the instructions on the regular exam.

Problem	Points	/out of
1		/ 16
2		/ 15
3		/ 15
Total		/ 46

Good luck!

Problem 1 Here's a data definition for representing M&M's: 16 POINTS

```
(define-struct m+m (kind color))
;; An M+M is a (make-m+m Kind Color)
;;
;; A Kind is one of:
;; - 'plain
;; - 'peanut
;;
;; A Color is one of:
;; - 'red
;; - 'yellow
;; - 'green
;; - 'blue
```

(a) Design the function `odd-plainblues?` that takes a list of `M+M`s and returns true if the list contains an odd number of plain blue M&M's. You must define the function using just `foldr`, as follows:

```
(define (odd-plainblues? mms)
  (foldr ...
```

You may use the following function:

```
(define (plain-blue? mm)
  (and (symbol=? (m+m-kind mm) 'plain)
       (symbol=? (m+m-color mm) 'blue)))
```

You may use the following list of M&M's in your tests:

```
(define mms1 (list (make-m+m 'plain 'red)
                   (make-m+m 'plain 'blue)
                   (make-m+m 'peanut 'yellow)
                   (make-m+m 'plain 'blue)
                   (make-m+m 'plain 'blue)))
```

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

(b) You also want to be able to determine how many yellow peanut M&M's will be left over after you take all the yellow peanut M&M's in a given list and evenly divide them amongst five people. As a good programmer, you know there's an opportunity for abstraction here!

Design a function `leftover` that takes a list of elements, a predicate `pred` on those elements, and a number `n`. The function should return the number of elements satisfying `pred` that are left over after dividing all the list elements satisfying `pred` into `n` equal sets.

You must define `leftover` using just `foldr`, as follows:

```
(define (leftover xs pred n)
  (foldr ...
```

Give `leftover` the most general contract possible.

Here are examples of how we expect to be able to use `leftover`:

```
(check-expect (leftover mms1 plain-blue? 2) 1)
(check-expect (leftover '(2 0 4 0 0) zero? 3) 0)
(check-expect (leftover '(2 0 4 0 1 0 0 0) zero? 3) 2)
```

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

(c) Define `odd-plainblues?` from part (a) again, this time using `leftover`. (There's no need to provide a contract, purpose statement, and tests again.)

Problem 2 All semester students have been asking us about objects, so we've decided to show you some on the exam. How would we represent objects in a functional langauge like ISL-λ? As functions, of course! For this problem you will implement a "*class*" of `Circle` objects. A `Circle` is an object-oriented (OO) representation of a circle, though you don't need to know *anything* about objects to do this problem; just pay careful attention to the description and the examples.

15 POINTS

Design a function `new-circle` that consumes two inputs, a `Posn` specifying the position of the center of the circle and a number representing the radius of the circle, and produces a `Circle`.

```
;; new-circle : Posn Number -> Circle
```

A `Circle` is a function that responds to *messages*. A message is sent by applying a `Circle` to a `Symbol` that matches the message's name. The object reacts by producing a value, which is frequently called a "method," that is, a function that will carry out some task on behalf of the object.
Here are the *contracts* of the messages your `Circle` representation must support:

Message Name	Message Result Contract
`'center`	`Posn`
`'radius`	`Number`
`'resize`	`[Number -> Circle]`
`'equal`	`[Circle -> Boolean]`

Sending a `Circle` the message `'center` (in other words, applying a `Circle` to the symbol `'center`) returns a `Posn` that represents the center of the circle (the first argument to `new-circle`); sending `'radius` returns the radius of the circle. Sending a `Circle` the message `'resize` returns a function that consumes a number indicating how much to change the radius by and constructs a new circle with the center unchanged and the radius increased by the given amount. Sending a `Circle` the message `'equal` returns a function that when applied to another `Circle` determines if the circles have the same centers and radii.

Hint: The next page contains some examples/tests to further clarify the details.

Task: Design `new-circle`.

```
;; Example Circles...
(define c0 (new-circle (make-posn 10 20) 4))
(define c1 (new-circle (make-posn 10 20) 9))

;; Tests for each 'message'
(check-expect (c0 'radius) 4)
(check-expect (* (posn-x (c0 'center))
                 (posn-y (c0 'center))) 200)
(check-expect (((c0 'resize) 10) 'radius) 14)
(check-expect ((c1 'equal) c0) false)
(check-expect ((((c1 'resize) -5) 'equal) c0) true)
```

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

Problem 3 An oracle is a function that knows about a number and can respond to guesses about the number. Here is our data definition for Oracles: 15 POINTS

```
;; An Answer is one of:
;; - 'low
;; - 'high
;; - 'ok
;;
;; An Oracle is a [Number -> Answer]
```

The oracle `wilma`, for example, knows about the number 4:

```
(wilma 2) ; produces 'low
(wilma 3) ; produces 'low
(wilma 4) ; produces 'ok
(wilma 5) ; produces 'high
(wilma 6) ; produces 'high
```

(a) Design a function `number->oracle` that makes an oracle for a given number.

(b) Design a function `oracle->number` that consumes an oracle and two integers, `lo` and `hi`, and produces the number the oracle knows. Assume that `lo < hi`, and that the number known to the oracle is an integer in the range `[lo,hi)`.

Your function must be efficient; it should only make at most about 20 guesses in order to find a number in the range [0,1000000).

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