CSU211 Exam 2 – Fall 2007

Name: _____

Student Id (last 4 digits): _____

Instructor's Name: _____

- 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.
- The phrase "design this function/program" means that you should apply the design recipe. You are *not* required to provide a template unless the problem specifically asks for one. Be prepared, however, to struggle with the development of function bodies if you choose to skip the template step.
- You may obtain a maximum of 55 points: 50 for the first six problems; and five extra-credit points for the final problem.
- 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.

Problem	Points	/out of
1		/ 6
2		/ 3
3		/ 13
4		/ 6
5		/ 9
6		/ 13
Extra		/ 5
Total		/ 55
Base	**50**	

Good luck!

Problem 1 6 POINTS

```
(define-struct student (name instructor awake? quizzes))

;;; A Student is (make-student String  Symbol
;;;                            Boolean [Listof Number])
;;; where: instructor - one of 'Olin or 'David
;;;        awake?     - true if the student asks
;;;                     and answers questions in class
;;;        quizzes    - the list of grades assigned
;;;                     for the class quizzes
```

Design a function, `instructor->students`, that takes an instructor and a list of students, and returns all the students in that instructor's section.

Problem 2 Design the function `prefix`, which takes a number n, and a list, and returns the list containing the first n elements of the input list. You may assume n is a non-negative integer less than or equal to the length of the input list.

3 POINTS

Problem 3 We order strings using *lexicographic order*, where, for example, 10 POINTS

$$\text{“cap”} < \text{“capillary”} < \text{“cats”}$$

This is how we order words in a dictionary.

We can apply the idea of lexicographic order to lists of numbers. A list of numbers a is *lexicographically less than* another list b if either (1) a is a proper prefix of b, or (2) some element of a is less than the corresponding element of b, and the two lists match identically on the preceeding elements. So, for example,

```
(lex< '(3 -8) '(3 -8 9 -100 5 7))
(lex< '(3 2) '(3 5))
(lex< '(3 -8 9 -100 5 7) '(3 -8 10 -500 -600 -999))
```

Design `lex<`, the comparison function that takes two lists of numbers as inputs, and determines if its first argument is lexicographically less than its second argument.

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

Problem 4 Rewrite these functions to use loop functions instead of explicit recursion. You may use `lambda` and `local` in your definitions. 13 POINTS

```
;;; contains? : [Setof Number] Number -> Boolean
;;; Is the number in the set?
;;; (We represent sets with lists of elements,
;;; repetition permitted.)
(define (contains? ys x)
  (cond [(empty? ys) false]
        [else (cond [(= x (first ys)) true]
                    [else (contains? (rest ys) x)])]))
```

```
;;; union : [Setof Number] [Setof Number] -> [Setof Number]
;;; Union two sets of numbers, where sets are represented
;;; as lists WITH NO REPETITIONS PERMITTED.
(define (union a b)
  (cond ((empty? a) b)
        (else (local ((define elt (first a))
                      (define u (union (rest a) b)))
                (cond ((contains? u elt) u)
                      (else (cons elt u)))))))
```

```
;;; increment-by : Number [Listof Number] -> [Listof Number]
;;; Increment every number in the list by the given amount.
(define (increment-by change nums)
  (cond [(empty? nums) empty]
        [else (cons (+ change (first nums))
                    (increment-by change (rest nums)))]))
```

Problem 5 What is the contract for this function? 6 POINTS

```
(define (count test xs)
  (cond [(empty? xs) 0]
        [else (local [(define n (count test (rest xs)))]
                (cond [(test (first xs)) (+ 1 n)]
                      [else n])))]))
```

Problem 6 You have a summer job developing code at a company that uses a cheap, cut-rate Scheme system, PowerSkeem!, which the owner's nephew wrote. The bad news is that PowerSkeem!, among many other problems, doesn't have the `map` or `filter` functions. The partial good news is that it *does* have a `foldr` function. 9 POINTS

It's going to take you some time to convince your boss to switch over to Dr. Scheme. In the meantime, you'd like to program using `map` and `filter`. So you need to define them first. Fortunately, you're sophisticated enough to realise that you can write both `map` and `filter` rather compactly using `foldr`.

Your task: Design `map` and `filter` using `foldr`. You may use `lambda` or `local`, if needed.

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

Problem 7 (Extra credit) 13 POINTS

Alas, the entire number system in our Scheme system appears to be deeply broken—adding 3 to 5 produces 8 sometimes, but other times, it produces 95. We're not entirely sure why. These things happen.

While we investigate the problem, we are working around it by defining our own number system, without using any of the built-in Scheme numbers.

We start with the following structure and data definition:

```
(define-struct double (n))
(define-struct doubleplus (n))

;;; A BNum is one of
;;; - 'zero                      (represents 0)
;;; - (make-double BNum)         (represents 2*n)
;;; - (make-doubleplus BNum)     (represents 1+2*n)
```

Thus, the BNum representing 3 is:

```
;;; 3 = 1 + 2 * (1 + 2 * 0)
(make-doubleplus (make-doubleplus 'zero))
```

- Write down the BNums for the numbers 0, 5 and 10.

- We hired a coop student to implement a full suite of numeric operations using BNums. She successfully implemented an addition operation,

  (b+ *bnum* *bnum*)

  and was starting on the multiply operation,

  (b+ *bnum* *bnum*)

  when she was tragically killed by a stray ricochet while driving golf balls in her dorm shower. We've hired you to finish the job (with the programming, that is, not the golf balls).

  Please design the function `b*` that will multiply two BNums. You may assume `b+` has already been written.