CSU2500 Exam 2 – Fall 2009

Name: ____________________

Student Id (last 4 digits): ____________________

Section (morning, honors or afternoon): ____________________

- 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.
- You may write $c \rightarrow e$ in place of `(check-expect c e)` to save time writing. You may also write the Greek letter λ instead of `lambda`, to save writing.
- 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		/ 5
2		/ 4
3		/ 6
4		/ 12
5		/ 8
6		/ 15
7		/ 15
Total		/ 55
Base	**50**	

Good luck!

Problem 1 Suppose we have the two lists 5 POINTS

```
(define a '(1 2))
(define b '((3 4) (5 6)))
```

What do each of the following expressions produce:

1. `(append a b)`
2. `(list a b)`
3. `(cons a b)`
4. `(apply append a b)`

Problem 2 The set operations we designed in class (`union`, `contains?`, *etc.*) only work when the set elements are things that can be compared using the `equal?` function. Let's design more general versions, where we can specify our own comparison function. Here's a data definition for a set: 5 POINTS

```
;;; A [Setof X] is a [Listof X].
;;; Repetitions not allowed.
```

Design a function `contains?` which tells if a set `s` contains some element `item`. Set elements are compared using the third agument to the function, `elt=?`.

For example, if we compare numbers by comparing their absolute values

```
;;; abs=? : Number Number -> Boolean
;;; Do the two numbers have the same absolute value?
(define (abs=? x y) (= (abs x) (abs y)))
(check-expect (abs=?  0  0) true)
(check-expect (abs=?  3  3) true)
(check-expect (abs=?  3 -3) true)
(check-expect (abs=? -3  3) true)
(check-expect (abs=? -3 -3) true)
(check-expect (abs=?  3  0) false)
```

then we should get the following behavior:

```
(contains? '(5 1 7)  1 abs=?) ; Should produce true.
(contains? '(5 1 7) -1 abs=?) ; Should produce true, too.
(contains? '(5 1 7)  8 abs=?) ; Should not produce true.
```

Write your function using a loop function.

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

Problem 3 We've lost confidence in Dr. Scheme's built-in `andmap` function. Please design a replacement, named `andmap2500`. (Since the contract and purpose statement can be taken from the course textbook, you only have to show the code and tests/examples.) 4 POINTS

Problem 4 You are working at the city zoo, writing software to help the zookeepers keep track of the animals. You employ the following data definition to represent the zoo's residents: 5 POINTS

```
;;; An ANIMAL is (make-animal String Number)
(define-struct animal (name legs))

(define a1 (make-animal "Mr. Ed" 4))
(define a2 (make-animal "Flipper" 0))
(define a3 (make-animal "Shelob" 8))
```

The zookeepers would like a function, `by-legs`, that takes a number of legs (such as 4) and a list of animals, and returns all the animals in the input list who have the given number of legs.

Design `by-legs`; use a loop function.

Problem 5 Here are two equivalent definitions of a function, `fred`. (You may work with whichever one seems clearer to you.) Provide a contract for the function.

6 POINTS

```
(define-struct pair (a b))

;;; A [Pair X Y] is a (make-pair X Y).

(define (fred test lop)
  (cond [(empty? lop) false]
        [(test (pair-a (first lop))) true]
        [else (fred test (rest lop))]))

(define (fred test lop)
  (and (not (empty? lop))
       (or (test (pair-a (first lop)))
           (fred test (rest lop)))))
```

Problem 6 We can make binary trees with strings for leaves, or binary trees with numbers for leaves, or binary trees with anything we'd like for the leaves, using the following data definition. 12 POINTS

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

;; A [BT X] is one of:
;; - an X
;; - (make-node [BT X] [BT X])
```

For example, we can make a binary tree of strings (*i.e.*, a `[BT String]`) with

```
(define bt1 (make-node (make-node "Olin" "Shivers")
                       (make-node "David"
                                  (make-node "Van" "Horn"))))
```

Recall that the `foldr` operation allows us to process the elements of a list: add them up, multiply them together, assemble them into a set, *etc.* Similarly, we can definine an analogous "loop function" to fold up a binary tree, called `fold-tree`. Applying `fold-tree` with these arguments

```
(fold-tree + string-length bt1)
```

will replace every occurrence of `make-node` in `bt1` with `+`, and every leaf `s` with `(string-length s)`, computing

```
(+ (+ (string-length "Olin") (string-length "Shivers"))
   (+ (string-length "David")
      (+ (string-length "Van") (string-length "Horn"))))
```

In other words, the first argument to `fold-tree` says what to do to the `node` structures of the tree, while the second argument says what to do to its leaves; in the example above, we produced the total length of all the strings in the tree.

1. Design `fold-tree`.
   (Hint: You might want to check your contract against the `bt1` example above.)

2. Use `fold-tree` to define the `height` function, which produces the height of a tree. (Assume the height of a leaf is zero.)

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

Problem 7 You may have heard of the `cons` function, which adds an element to the front of a list. Less well known is the `snoc` function, which adds an element to the *end* of a list: 5 POINTS

```
(snoc 7 '(1 3 5)) ; produces '(1 3 5 7)
```

Define `snoc` using a loop function.

Problem 8 Refer back to problem 6 for the `BT` data definition. 15 POINTS

```
;;; A NumTree is a [BT Number].
```

Design the function `same-shape?` that determines if two NumTrees have the same shape—that is, we ignore the actual numeric values at the leaves, and produce true if the first tree has a `node` structure everywhere the second tree does, and a leaf everywhere the second tree does.