CSU2500 Exam 2 – Fall 2010

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` → `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		/ 16
2		/ 13
3		/ 8
4		/ 6
5		/ 13
Extra		/ 6
Total		/ 62
Base	**56**	

Good luck!

Going Postal

The Programming Research Lab (PRL) at Northeastern, which includes the 2500 professors and many of your TAs, was invited for a field trip to Google's lab in Kendall Square today. Russ Cox, the guy who led the design and implementation of Google Code Search, asked the PRL team to come have lunch, meet with some engineers, and pitch their new project: a next generation webmail program the PRL hopes will replace GMail.

Unfortunately, on the night before the visit, many of the PRL team were preoccupied writing an exam for the intro to programming course. As a consequence, they ran out of time to finish their prototype of the "NUPostal" mail system. (If only they had started earlier!) Not wanting to miss the chance to write the successor to one of the world's most widely used web applications, or a free lunch, the PRL decided to show Russ and the Google engineers their data definitions and describe the design, but explained they'd have to send in the actual program later. Russ liked the data definitions and agreed to let the PRL team submit their program later; but the enginners would need to start reviewing the program no later than 9 P.M. that night. As Russ walked the group back to the red line, he said NUPostal could go live as early at 2 A.M., assuming the program was well designed.

Excited by the prospect, the PRL team came back to Northeastern, but quickly realized there was no way they could finish their NUPostal program *and* administer the exam. But then, in a moment of inspired deliberation on how to get out of this pickle, it came to them: let's have you do it! (Twice!)

The next two problems ask you to develop key components of the NUPostal system: once with structural recursion, and once with loop functions.

Problem 1 Here are the data definitions pitched to Google: 16 POINTS

```
;; A MailBox is a [Listof Email].

;; An Email is a (make-email Address String String).
(define-struct email (from subject body))

;; An Address is a Symbol. For example, 'wizwoz@foobar.com
```

Important: For this problem, you *must* use the design recipe for structural recursion (in particular, you should *not* use loop functions; for that, see problem 2).

A) Design the program `emails-from` that, given a mailbox and an email address, computes the list of emails that are from the given address in the mailbox.

B) Design the program `subjects` that computes the list of email subjects in a given mailbox.

C) Design the program `total-size` that computes the total size of a given mailbox in terms of the length of all the email bodies in it. Again, do *not* include the size of the from address or the subject line in your total—just the message bodies.

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

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

Problem 2 The programs of the previous problem, if developed with the design recipe for structural recursion, should follow patterns abstracted by one of the loop functions. For each of the programs you wrote, re-write it using a loop function. In each case, *one* loop function should suffice. 13 POINTS

Since you have already written the contract and purpose statement (right? it is Google policy, after all) you *only* need to write the function definitions.

A) Write `emails-from` using a loop function.

B) Write `subjects` using a loop function.

C) Write `total-size` using a loop function.

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

Problem 3 Design a function, `weaver`, that weaves two lists into a single list by alternating their elements. Make sure you give it a general contract. 8 POINTS

Here are some tests/examples to further explain:

```
(check-expect (weaver empty (list 1)) (list 1))
(check-expect (weaver (list "Hi" "Lo") empty) (list "Hi" "Lo"))
(check-expect (weaver (list 1 2) (list 3 4))
              (list 1 3 2 4))
(check-expect (weaver (list 'R 'C 'R) (list 'A 'E))
              (list 'R 'A 'C 'E 'R))
```

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

Problem 4 Here is a definition of a function, `do-while`, that does something. Study the definition, and give it a general contract. 6 POINTS

```
(define (do-while ys go what)
  (cond [(empty? ys) empty]
        [(not (go (first ys))) empty]
        [else
          (cons (what (first ys))
                (do-while (rest ys) go what))]))
```

Problem 5 You've seen lists built with `cons` throughout the semester, where elements are added onto the *front* of a list. Less well known are `snoc` lists, where elements are added onto the *end* of a list. 13 POINTS

```
;; A [SnocListof X] is one of:
;;  - empty
;;  - (make-snoc [SnocListof X] X)
(define-struct snoc (front last))
```

For instance, we would represent the list `'(1 2 3)` as:

```
(make-snoc (make-snoc (make-snoc empty 1) 2) 3)
```

A new social-media/messaging company, Blather, uses this data format extensively, and wants to be able to use the familiar loop functions, even with their backwards lists.

Help them by designing the abstract loop function, `snoc-filter`, which takes a predicate and a `[SnocListof X]` and returns a `snoc` list that contains only the elements for which the predicate returns `true`.

Here's a couple examples/tests to clarify:

```
(define slst (make-snoc (make-snoc (make-snoc empty 1) 2) 3))
(check-expect (snoc-filter odd? slst)
              (make-snoc (make-snoc empty 1) 3))
(check-expect (snoc-filter even? sl)
              (make-snoc empty 2))
```

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

Problem 6 (Extra credit) 6 POINTS

Shivers and Van Horn have been acting suspicious lately, and Chadwick suspects that they've been planning something behind his back. He noticed that they've been passing "*secret*" messages. And the message format? Of course they're using lists of `Numbers`! He's intercepted a few of their messages, and thinks that the secret code is based on binary trees of `Strings`.

Here's his data-definition:

```
;; A CodeTree is one of
;;  - String
;;  - (make-node CodeTree CodeTree)
(define-struct node (zero one))
```

The idea is to encode each letter of the message as a `Number` that represents a path from the root of a `CodeTree` to a `String` at a leaf, where going to the left (`zero`) means the number is even, and going right (`one`) means the number is odd.
For example, here's a tree that encodes `"a"` as the number `0`, and `"b"` as the number `1`:

```
(make-node "a" "b")
```

Of course, Shivers and Van Horn are smart, so they've encoded using a *recursive* structure in order to have more than just 2 letters. For instance, the following tree encodes `"a"` as `0`, `"b"` as `2`, `"c"` as `1`, and `"d"` as `3`:

```
(make-node (make-node "a" "b")
           (make-node "c" "d"))
```

Note that (i) the deeper the tree, the longer the numbers, (ii) the encodings of `"a"` and `"b"` are the ones from before multiplied by `2`, and (iii) the encodings of `"c"` and `"d"` can be seen as `(1+0)` and `(1+2)` respectively.

Your task: write a function, `message`, that accepts a `[Listof Number]` and a `CodeTree`, and returns the corresponding decoded message. *Hint*: it's helpful to design a function `decode` that decodes a single `Number` into its corresponding `String`, given a `CodeTree`.

The next page contains some helpful tests/examples for both functions.

```
(define t-1 (make-node "m" "o"))
(define t-2 (make-node (make-node "w" "t")
                       (make-node "e" "s")))

(check-expect (decode 0 t-1) "m")
(check-expect (decode 0 t-2) "w")
(check-expect (decode 3 t-2) "s")

(check-expect (message '(0 1 0) t-1) "mom")
(check-expect (message '(0 1 1) t-1) "moo")
(check-expect (message '(3 0 1 1 2) t-2) "sweet")
(check-expect (message '(2 0 1 1 2) t-2) "tweet")

(check-expect (message '(0 7 3 1 6 2 0 5)
                       (make-node
                        (make-node "c" (make-node "i" "w"))
                        (make-node (make-node "d" "k")
                                   (make-node "a" "h"))))
              "chadwick")
```

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