CSSE 304 Exam #2 Part 4, April 26, 2015 (day 28b) Na _____ ...sion (circle one) 01 (02) 03

Problem	Possible	Earned	Comments
C1 – vector-list	50	50	
C2 - when	25	25	
Total	39	85	
C1 Extra	10	10	

During the entire exam you may not use email, IM, cell phone, PDA, headphones, ear buds, or any other communication device or software. Efficiency and elegance will not affect you score, provided that I can understand your code.

Part 4, programming. For this part, you may additionally use code a Scheme editing/programming environment, the textbooks from the course, The *Chez* Scheme Users' Guide, the PLC grading program, and any materials that I provided online for the course. You may not use any other web or network resources. You are allowed to look at and use any Scheme code that *you* have previously written.

You may assume that all input arguments will be of the correct types for any procedure you are asked to write; you do not need to write code to check for illegal input data. Of course you are allowed to use mutation for this problem, since the very purpose of several parts involves mutation. **Test your Part 3 code off-line before you submit to the grading server** (an attempt to reduce the load on the server). **Remember!** Code submitted to the server must not include any output-producing statements, e.g., `display`, `write`, or `trace`.

C1. (50 points) In this problem, you will write code to produce `vector-list` "objects", in a style similar to the `stack` and `iterator` objects. They provide part of the functionality of Java's `ArrayList` objects. A `vector-list` 's elements are stored in a Scheme vector.
Two additional quantities are needed, representing the *size* and *capacity* of the `vector-list` (these quantities have the same names in the Java 7 documentation of the `ArrayList` class). *Capacity* is the length of the vector that holds the list elements. *Size* is the current length of this object's list. . If *size* = *capacity*, and we want to add another element to the list, we must allocate a new vector, copy elements from the old vector to the new vector, adjust any other "fields" that need to be adjusted, and add the new element to the new vector. In CSSE 230, we show that whenever we increase the capacity in this way, doubling the size is a good strategy. Of course *size* ≤ *capacity* must always be true
In Java's `ArrayList` class, the user cannot see the underlying array or its capacity. To assist me in testing your code, your `vector-list` must have a `capacity` method. For ease of debugging, you may want to also add a method that shows the current value a `vector-list` 's vector.

The `vector-list` constructor takes a vector as its argument. It produces a `vector-list` whose vector representation is a copy of the original vector, and whose capacity and size are both initially equal to the length of that vector.

The **transcript on the back of this page** should indicate how this all works.

10 points extra-credit. Write your code so that within the implementation of any method you can use `this` to refer to the current object. Example: `(this 'capacity)`.

C2. (25 points) Enhance your interpreter so that it implements *Chez* Scheme's `when` syntax. Here is the description of `when` from *The Chez Scheme User's Guide*:

syntax: `(when test-expr expr1 expr2 ...)` For `when`, if *`test-expr`* evaluates to a true value, the expressions $expr_1$ $expr_2$ ... are evaluated in sequence, and the value of the last expression is returned. If *`test-expr`* evaluates to false, none of the other expressions are evaluated, and the value of `when` is unspecified.

Note: to get that unspecified behavior, simply use "one-armed if (no "else" part) in the implementation of `when`.

See examples on the back of this page.

```
> (define v '#(4 5 6)) ; make a vector
> (define vl  (vector-list v)) ; make a vector-list
> (define vl2 (vector-list v)) ; and another
> (vl 'size)
3
> (vl 'get 2)
6
> (vl 'set 2 'a)
> (vl 'get 2)
a
> (vl2 'get 2) ;different than previous answer.
6
> (vl 'add 'f) ; adds at end,
               ; doubles vector length if necessary
> (vl 'remove) ; removes last element
f
> (vl 'add 'xyz)
> (vl 'add 'abc)
> (vl 'capacity)
6
> (vl2 'capacity)
3
>
```

```
> (list
  (eval-one-exp '
   (let ([a 3] [b (list 4)])
     (when (< a 5)
       (set-car! b (+ (car b) a))
       (set-car! b (+ (car b) a)))
     b))
  (eval-one-exp '
   (let ([a 3] [b (list 3)])
     (when (> a 5)
       (set-car! b (+ (car b) a))
       (set-car! b (+ (car b) a)))
     b)))
((10) (3))
>
(list
 (eval-one-exp '
  (let ([a 3] [b (list 4 5)])
    (when (< a 5)
      (set-car! (cdr b) (+ (car b) a))
      (set-car! b (+ (car b) a)))
      (set-car! b (+ (car (cdr b)) a))
    b))
 (eval-one-exp '
  (let ([a 3] [b (list 3 4)])
    (when (> a 5)
      (set-car! (cdr b) b (+ (car b) a))
      (set-car! b (+ (car b) a)))
    b)))
((10 7) (3 4))
```