A. Short Answer

1. When evaluating an expression like ((\(\lambda\) \(x\) \((+ x 2)\)) 3), we say that \(x\) becomes **bound** to 3. How is this different from saying that \(x\) is **assigned** the value 3?
   (a) There is no difference.
   (b) Variable \(x\) represents 3 only in the context of \((+ x 2)\). Once this expression has been evaluated and the value returned, \(x\) resumes its former value.
   (c) A variable can be bound to only one value, while it can be assigned many different values.
   (d) Bound values are not shadowed by further bindings, while assignments are overwritten by later assignments.


2. The result of evaluating ((\(\lambda\) \(x\) ((\(\lambda\) \(y\) \((+ x y)\) 4)) 5)) is a procedure (T/F)


3. The result of evaluating (map (\(\lambda\) \(f\) \(f\)) (map (\(\lambda\) \(x\) (\(\lambda\) \(y\) \((+ x y)\)) '(1 2 3 4))) is a procedure (T/F)


4. The result of evaluating ((\(\lambda\) \(x\) \(x\)) (\(\lambda\) \(x\) \(x\))) is a procedure (T/F)


5. Which of the following is true about the expression (map \(f\) 1)?
   (a) It returns \(f\) 1.
   (b) It requires \(f\) to be a function of 2 arguments.
   (c) The list 1 must be nonempty.
   (d) It returns a list in which each element is the corresponding element of 1 transformed by \(f\).


6. According to the Scheme standard, the order of evaluation of function arguments in Scheme is:
   (a) always from left to right.
   (b) always from right to left.
   (c) not specified, and thus may vary from one implementation to another.
   (d) guaranteed never to affect the value of the function’s result.


B. **Medium Answer** – Answer each of the following as instructed. Some partial credit may be given.

7. Draw the consbox diagram for ((a . b) (c d (e f)))
   Example: the consbox diagram for (a b c) is
   ![Diagram]
   
8. For each variable in the following Scheme expressions
   - Circle each bound occurrence and draw an arrow to the entry in the appropriate lambda binding list that declares the variable.
   - Underline each free occurrence.
   
   For example:
   
   (lambda (x) (lambda (y) (e x (f y z))))

   (a)  (lambda (cdr)
         (car (lambda (x) (cdr x))))

   (b)  ((lambda (l)
         ((lambda (f)
           (if (null? l)
               (f (lambda (x y) (x l)) x)
               (f (lambda (f 1) (f (f 1))) l)))
             (lambda (g l) (cons (car l) (g f (cdr l)))))))
       '((a b c d e))
C. **Streams** This question concerns streams. You may assume the existence of Scheme functions `cons$` `car$` `cdr$` as given in class and lab.

(a) What is a data *stream*? Answer this question without any mention of Scheme models. Give an example of a "real life" data stream that programmers may need to deal with.

(b) How have we modeled streams in Scheme? Answer this question with specific details about the implementation of streams in Scheme.

(c) Write a function `merge$` to create a new stream from two given streams by alternately taking its members from one and then the other given stream. Thus

```
(merge$ odd-ints even-ints)
```

produces the stream \(1, 2, 3, 4, 5, 6, \ldots\).

(d) What are the first six numbers in the stream `mystery` where:

```
(define add$ (lambda (stream1 stream2)
  (cons$ (+ (car$ stream1) (car$ stream2))
    (add$ (cdr$ stream1) (cdr$ stream2)))))

(define mystery (cons$ 11 (cons$ 25 (add$ mystery (cdr$ mystery))))))
```