1. Suppose $a$, $b$, $c$ and $d$ are defined as $\#t$. What would be the result of evaluating the following expressions (true or false)?

   i. $(\text{and} \ (\text{not} \ a) \ (\text{or} \ b \ c \ d)) \rightarrow \text{false}$

   ii. Suppose $a$, $b$, $c$ and $d$ are defined as $\#f$. What would be the result of evaluating the following expression (true or false)?

   $$(\text{or} \ a \ (\text{not} \ (\text{not} \ b)) \ (\text{not} \ (\text{and} \ c \ d))) \rightarrow \text{true}$$

   iii. Circle any parts of the two expressions above that are NOT evaluated with this set of inputs. If you think no parts are unevaluated, write "all evaluated."

   In part i, the $(\text{or} \ b \ c \ d)$ would not be evaluated due to the short-circuit evaluation done by the and special form. The last $d$ at the end of the expression in part ii) is not evaluated due to the short-circuit evaluation done by the and special form.

2. Consider the following 3 example lists: EVENS, ODDS, and THREES.

   ;; Examples
   (define EVENS (cons 2 (cons 4 (cons 6 (cons 8 (cons 10 empty))))))
   (define ODDS (cons 1 (cons 3 (cons 5 (cons 7 (cons 9 empty))))))
   (define THREES (cons 3 (cons 6 (cons 9 (cons 12 (cons 15 empty)))))

   (a) Show the result of evaluating the following expressions, keeping in mind that the evaluation may result in an error. If you think the statement will cause an error, just write ERROR for the result of evaluation.

   $$(\text{first} \ (\text{rest} \ \text{THREES})) \Rightarrow 6$$
   $$(\text{rest} \ (\text{rest} \ (\text{first} \ \text{ODDS}))) \Rightarrow \text{ERROR}...\text{rest cannot be used on a non-list}$$
   $$(\text{append} \ \text{ODDS} \ (\text{list} \ 11)) \Rightarrow (\text{list} \ 1 \ 3 \ 5 \ 7 \ 9 \ 11)$$
   $$(\text{rest} \ (\text{rest} \ (\text{rest} \ \text{EVENS}))) \Rightarrow (\text{list} \ 10)$$

   (b) Write expressions using only first, rest, and/or cons on the lists defined above to produce the lists or list elements specified below.

   – Write an expression that evaluates to the list containing the last two numbers from the EVENS list.

     $$(\text{rest} \ (\text{rest} \ (\text{rest} \ \text{EVENS})))$$

   – Write an expression that evaluates to the list containing the first number of each list (i.e., the list containing 2, 1, 3).

     $$(\text{cons} \ (\text{first} \ \text{EVENS}) \ (\text{cons} \ (\text{first} \ \text{ODDS}) \ (\text{list} \ (\text{first} \ \text{THREES}))))$$
     or

     $$(\text{cons} \ (\text{first} \ \text{EVENS}) \ (\text{cons} \ (\text{first} \ \text{ODDS}) \ (\text{cons} \ (\text{first} \ \text{THREES}) \ \text{empty})))$$

   – Write an expression that evaluates to the list containing the first element of EVENS, the second element of ODDS, and the third element of THREES (i.e., the list containing 2, 3, 9).

     $$(\text{cons} \ (\text{first} \ \text{EVENS}) \ (\text{cons} \ (\text{first} \ (\text{rest} \ \text{ODDS})) \ (\text{list} \ (\text{first} \ (\text{rest} \ (\text{rest} \ \text{THREES}))))))$$
     or

     $$(\text{cons} \ (\text{first} \ \text{EVENS}) \ (\text{cons} \ (\text{first} \ (\text{rest} \ \text{ODDS})) \ (\text{cons} \ (\text{first} \ (\text{rest} \ (\text{rest} \ \text{THREES})))) \ \text{empty})))$$
3. Consider the define statements and representation of the Global Environment given below:

```
(define A 3)
(define B 2)
(define C (* A B))
(define X (+ C 3))
(define (fun x y) (* x (+ y 4)))
```

(a) Show, in the table below, the additions to the global environment created by the 5 definitions above when they are executed in DrRacket (assume entries are added from the top of the table downward).

<table>
<thead>
<tr>
<th>Global Environment</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>fun</td>
<td>(lambda (x y) (* x (+ y 4)))</td>
</tr>
</tbody>
</table>

(b) Using the Global Environment from part (a), show the result of evaluating each of the following expressions, keeping in mind that the evaluation may cause an error message (the > is the Interactions Window prompt). If you think the expression causes an error, just write ERROR.

```
> (expt C 2)
36

> (* B 1000)
2000

> (string-append A B)
ERROR: string-append: expects a string as 1st argument, given 3

> (string? X)
false

> (fun A X) => (lambda (3 9) (* 3 (+ 9 4))) => (* 3 13) =>
39
```

4. Write a function called `swapper`. This function should consume two strings, `str1` and `str2`, and a list of strings called `los`. The function should return a list of strings in which every instance of `str1` has been replaced by `str2` and every instance of `str2` has been replaced by `str1` in `los`. Write this function in 2 versions, one using regular recursion and the other using an accumulator.

```scheme
; Contract: (swapper str1 str2 los) -> list of strings
; Input: str1,str2:strings; los:list of strings
; Purpose: Replace str1 with str2 and str2 with str1 everywhere
; they appear in los

; Pre-function tests:
(check-expect (swapper "a" "b" (list "a" "b" "a" "b" "b")) (list "b" "a" "b" "a" "a"))
(check-expect (swapper "a" "b" empty) empty)
(check-expect (swapper "cat" "dog" (list "my" "dog" "has" "cat" "fleas")) (list "my" "cat" "has" "dog" "fleas"))
```
(define (swapper str1 str2 los)
  (cond
    ;; base case: los empty, return empty list
    [(empty? los) empty]
    ;; recursive case 1: first on los is same as str1, cons
    ;; str2 onto list returned by call to swapper on rest of los
    [(string=? str1 (first los)) (cons str2 (swapper str1 str2 (rest los)))]
    ;; recursive case 2: first on los is same as str2, cons
    ;; str1 onto list returned by call to swapper on rest of los
    [(string=? str2 (first los)) (cons str1 (swapper str1 str2 (rest los)))]
    ;; recursive case 3: first on los is not equal to str1 or str2,
    ;; cons first of los onto onto list returned by call to swapper
    ;; on rest of los
    [else (cons (first los) (swapper str1 str2 (rest los)))]))

; Contract: (swapper str1 str2 los) -> list of strings
; Input: str1,str2:strings; los:list of strings
; Purpose: Replace str1 with str2 and str2 with str1 everywhere
; they appear in los

; Pre-function tests:
(check-expect (swapper "a" "b" (list "a" "b" "a" "b" "b"))
  (list "b" "a" "b" "a" "a"))
(check-expect (swapper "a" "b" empty) empty)
(check-expect (swapper "cat" "dog" (list "my" "dog" "has" "cat" "fleas"))
  (list "my" "cat" "has" "dog" "fleas"))

; Function definition: (accumulator version)
(define (swapper str1 str2 los)
  (local
    ;; note: no need to pass str1 and str2 into accumulator
    ;; function because they do not change.
    [(define (swapper-acc los acc)
        (cond
          ;; base case: los empty, return empty list
          [(empty? los) acc]
          ;; recursive case 1: first on los is same as str1, call
          ;; swapper-acc tail-recursively and append acc onto list
          ;; created from str2
          [(string=? str1 (first los)) (swapper-acc (rest los) (append acc (list str2)))]
          ;; recursive case 2: first on los is same as str2, call
          ;; swapper-acc tail-recursively and append acc onto list
          ;; created from str1
          [(string=? str2 (first los)) (swapper-acc (rest los) (append acc (list str1)))]
          ;; recursive case 3: first on los is not equal to either
          ;; str1 or str2, call swapper-acc tail-recursively and
          ;; append acc onto list created from (first los)
          [else (swapper-acc (rest los) (append acc (list (first los)))]
        ))
    ]
    (swapper-acc los empty)))
5. Parts (a) and (b) refer to the mystery? function:

\[
\text{(define (mystery? x)}
\begin{align*}
& \text{(or (empty? (rest x))} \\
& \quad (\text{and (> (first (rest x)) (first x))}) \\
& \quad \text{(mystery? (rest x))})
\end{align*}
\]

(a) Show what the function mystery? returns for the following inputs or indicate that the function call would return an error by writing ERROR:

\[
\begin{align*}
\text{(mystery? (list 2 5 9 17))} & \quad \text{................ #t} \\
\text{(mystery? (list 1 2 5 4))} & \quad \text{............. #f} \\
\text{(mystery? (list 8))} & \quad \text{.................... #t} \\
\text{(mystery? (list 8 8 8 8))} & \quad \text{............. #f} \\
\text{(mystery? empty)} & \quad \text{...................... ERROR} \\
\text{(mystery? (list 1 2 2 4))} & \quad \text{............. #f}
\end{align*}
\]

(b) Clearly, the mystery? function consumes a flat list of numbers. But what else must we assume about the list mystery? consumes as input to ensure calling the function with that list as an argument does not produce an error? If you think the function works for any size flat list of numbers, just write “no further assumptions are necessary.”

The list consumed by mystery? must contain at least one number in order to not return an error. The function returns true only if there is a single number on the list or if all the numbers on the list are in strictly increasing order.

6. (6 points) Answer the following questions about lists.

(a) (2 points) Use cons and/or empty to construct the equivalent of the following lists:

List 1: (list 0 1 2 3 4 5)

Answer 1: \((\text{cons 0} \ (\text{cons 1} \ (\text{cons 2} \ (\text{cons 3} \ (\text{cons 4} \ (\text{cons 5 empty})))))))\)

List 2: (list)

Answer 2: empty

(b) (2 points) Use list to construct the equivalent of the following lists:
List 1: (cons 5 (cons 4 (cons 3 (cons 2 (cons 1 empty)))))

Answer 1: (list 5 4 3 2 1)

List 2: (cons empty empty)

Answer 2: (list (list))

(c) (2 points) Write the lists returned by the following expressions, using the list function in your answers:

Expression 1: (list (symbol=? 'a 'b) (symbol=? 'c 'c) false)
Answer 1: (list #f #t #f)

Expression 2: (list (+ 10 20) (* 10 20) (/ 10 20))
Answer 2: (list 30 200 0.5)

(d) (2 points) Write the lists returned by the following expressions, using the apostrophe notation and NOT the list or cons functions in your answers:

Expression 1: (list (symbol=? 'a 'b) (symbol=? 'c 'c) false)
Answer 1: '(#f #t #f)

Expression 2: (list (+ 10 20) (* 10 20) (/ 10 20))
Answer 2: '(30 200 0.5)
7. (5 points) Give the result of calling the following function with input (secret 4)

```
(define (secret x)
  (cond
   [(= x 0)
    (begin
     (printf "\nbase case, x = ~a\n" x)
     0)]
   [(= x 1)
    (begin
     (printf "\nbase case, x = ~a\n" x)
     1)]
   [else
    (begin
     (printf "\nrecursive case, x = ~a\n" x)
     (+ (secret (- x 2)) (secret (- x 1))))])

Solution for (secret 4):
recursive case, x = 4
recursive case, x = 2
base case, x = 0
base case, x = 1
recursive case, x = 3
base case, x = 1
recursive case, x = 2
base case, x = 0
base case, x = 1
3
```