CISC280 Midterm Learning Experience Practice 2

NAME _____

General instructions:

Turn off all noise-making electronic devices, such as cell phones. Disturbance by such a device during the exam may result in a penalty not to exceed a full letter grade.

You may leave the classroom once the exam has begun, but you may not return.

There are pages worth a total of points, and an extra credit problem worth 3 points. Read the problems very carefully. Identify what kind of answer the problem asks for. If writing a procedure, carefully look at requirements for input and output, and any restrictions on how it must be written or which other procedures may be used.

You may assume a list argument will be flat unless it is otherwise specified, and that the elements will not produce errors for the procedures described.

Do not do unnecessary testing. For example, testing for both list? and null? instead of using one test and then else would be considered unnecessary testing.

Do not make code unnecessarily inefficient. An extra procedure call here or there is ok, but do not make an $O(n)$ problem into $O(n^2)$.

Do problems you are confident about first. If you finish the problems you know, write what you do know about other problems to gain partial credit; but erroneous information may detract from that credit, so don't make stuff up.

You may use any of the Scheme primitives we use in class.

1. (5 pts) Define two lists and draw them using box and pointer notation. Then show the box and pointer list that would result from joining the lists with `append`.

2. (10 pts) Write a procedure adjoin for a binary search tree. Explain the big O of the procedure you wrote.

3. (10 pts) Write the procedure **intersection** for an ordered list representation of a set.

4. (6 pts) Define a fcn last-pair that walks through a non-nested proper list and returns the last cons pair.
   What is the big O? Explain.

5. 14 pts. Assume *all* the following expressions are evaluated **in order**. Fill in the blanks with the value or message printed by the interpreter. If there is an error or other Scheme message, you do not have to be exact.

```
> (define (f x)
     (set! x (/ x 2))
     x)
> (define a 80)
> (f a)

1a. _____

> (f a)

1b. _____

> (define b 50)
> (define (g x)
     (/ x 2)
     x)
> (g b)

1c. _____

> (g b)

1d. _____

> (define (h x)
     (define (k)
        (set! x (/ x 2))
        x)
   k)
> (h 100)

1e. _____

> (define k (h 100))
> (k)

1f. _____

> (k)

1g. _____
```

6. (20 pts) What is O() for:

7. (20 pts) Show a let statement for two local "variables" and then show the equivalent procedure (the procedure that underlies "let").

8. (20 pts) Define data abstraction. Describe reasons to include it in software design.

9. (30 pts) Compare and contrast data-directed programming, message-passing, and dispatching-on-type. Explain their advantages and disadvantages when used to implement generic operations. In particular, what transactions are relatively easy or hard, and why?

10. (20 pts) Write code to simulate a bank account with features for deposit, withdraw, and password checking. Account variables must be local.

11. (10 pts) Consider the problem "Give the big O (time) for union of two sets." Is this question meaningful? Explain. How would you go about answering it?

12. (10 pts) Comment on this quote from the GNU C tutorial:

    > The assignment operator: No operator such as addition (+) or multiplication (*) would be useful without another operator that attaches the values they produce to variables. Thus, the assignment operator = is perhaps the most important mathematical operator.

13. (26 pts) Show what is returned by the interpreter in response to each of these expressions. Each is to be evaluated by itself. If the result is an error, or other interpreter message, you do not have to be exact. For example, if the expression were the input "expt", you could show the interpreter's response as "primitive" instead of writing #<primitive:expt>. If you write more than one thing after a question **CIRCLE** the one you want graded.

    (a) `(cons 3 4)`

    (b) `(cons 8 ())`

    (c) `(list 8 ())`

    (d) `(append () ())`

    (e) `(cons (list 1 2)(list 3 4))`

    (f) `(list (list 1 2)(list 3 4))`

    (g) `(append (list 1 2)(list 3 4))`

    (h) `(cdr (cons 3 ()))`

    (i) `(if (eq? (list 1) (list 1)) 2 3)`

    (j) `(quote a)`

    (k) `(cons () '(1))`

    (l) `((lambda (x) (+ x x)))`

    (m) `((lambda (x) (* x x 3) 4) 5)`

14. (7 pts) Draw the box and pointer notation for (2 ((3 (4)) 5) 6)

15. (5 pts) Write a **call** to your sequence toolkit that generates the matrix shown. You may use lambdas, but do not separately define procedures.

```
((0 1 2 3 4) (5 6 7 8 9) (10 11 12 13 14) (15 16 17 18 19))
```

16. (5 pts) What is big O for time of f1? Explain. (Remember that you aren't as concerned with the result of this call as you are with the behavior of the process.)

```
(define (f1 alist)
  (accumulate (lambda (a b) (+ (length alist) b))
              0
              alist))
```

17. (5 pts) What is big O for time of f2? Explain.

```
(define (f2 alist)
  (accumulate (lambda (x y) (cons (fast-expt 2 x) y))
              ()
              alist))
```

18. (5 pts) Consider the following two procedures. Xenon thinks that one of these should be faster than the other, but isn't sure which one. Help Xenon decide, and briefly explain your answer.

```
(define (one alist)
  (let ((x (length alist)))
    (* x x)))
```

```
(define (two alist)
    (* (length alist)(length alist)))
```

19. (5 pts) Define a function **make-multiplier** that takes a single factor as a parameter and returns a procedure that multiplies its argument by that factor:

```
> ((make-multiplier 3) 3)
9
>
```

(a) (3 pts) Now make a single call to a sequence tool that will use make-multiplier, and define **x** to be a list of multipliers that use the factors in the list `(2 3 4 5)`. After the call x will be a list of procedures.

```
(define x
```

(b) (3 pts) Now use the list of multipliers, x, and a single call to a sequence tool to make a list of the results of calling each multiplier on the number 3, so the result will be `(6 9 12 15)`. You will need a lambda function.

```
> _____________________________________________________
```

20. (5 pts)In class we defined a procedure cons. Fill in the three blanks:

```
(define cons

  _____________________________
   (lambda (msg)
     (cond ((equal? msg 'car) ________)

           ((equal? msg 'cdr) ________)
           (else 'error)))))
```

21. (10 pts) Xoonar, trying to attract the attention of Potstkr, writes union for **binary search trees**. Xoonar's code traverses the first tree (size n), calling adjoin to add each element to the second tree (size m). Assume the resulting tree is fairly well balanced. What is big O? Explain why very briefly.

    Potstkr is not impressed. Xoonar asks for your help. Assume you have access to your textbook. Explain to Xoonar a better way to write union, being **specific** about what procedures you would use and why it would be better.

22. (10 pts) Why use data abstraction in software design?

23. (10 pts) You are managing seven hospital databases as part of a single system, and you need to choose a method (one discussed in class) for implementing generic operations. Suppose that you think every hospital will need to perform essentially the same operations on records, and you expect new hospitals (with new record structures) to join your system. Which method would you choose, and why? Answer in terms of the information given here. Be concise.

24. (5 pts) Which technique for implementing generic operations requires all procedures in the system to have different names?

25. (5 pts) Which technique for implementing generic operations can be viewed as a "row" of a different technique?

26. (10 pts) Suppose we want to put an operation **add** into a table (as discussed in class). We want the operation to work on exactly two numeric arguments. Any number can be either 'rectanglar or 'polar type. What do we need to do to make this work correctly *without* implementing coercion? Be specific. A small drawing could help.

27. (5 pts) Consider the problem raised in number . How might this be handled in a message passing system? Think before answering.

28. (5 pts) In the rational number system data abstraction example, we discussed whether gcd should be used inside the constructor (make-rat) or inside the selectors (numer and denom). Do not tell me which. How was this discussion related to data abstraction?

29. (10 pts) Consider the code below. Rewrite any procedures that do not use excellent data abstraction as shown in the text and discussed in class. Only address data abstraction issues.

```
;Rectangular representation of complex numbers

(define (real-part z) (car z))

(define (imag-part z) (cdr z))

(define (magnitude z)
  (sqrt (+ (square (car z)) (square (cdr z)))))

(define (angle z)
  (atan (cdr z) (car z)))

(define (make-complex-from-real-imag x y) (cons x y))

(define (make-complex-from-mag-ang r a)
  (cons (* r (cos a)) (* r (sin a))))

(define (add-complex z1 z2)
  (make-from-real-imag (+ (car z1) (car z2))
                       (+ (cdr z1) (cdr z2))))
```

30. One simple way to distinguish between different representations of one kind of data is to attach a distinguishing symbol to each instance of the data. This is called

    ``_____''.

31. After attaching the symbol, a function that examines the symbol and calls the appropriate function on the data is said to be

    ``_____''.

32. The strategy described above allows the creation of generic procedures, procedures which can transparently operate on multiple representations of the same data. Under what circumstance does this strategy require every generic procedure that has been written so far to be modified? A short phrase is sufficient.

```
(define (apply-generic op . args)
  (let ((type-tags (map type-tag args)))
    (let ((proc AAA
      (if proc
          (BBB CCC (map contents args))
          (error
            "No method for these types -- APPLY-GENERIC"
            (list op type-tags))))))
```

Consider this incomplete code.

33. Show the scheme expression that should replace AAA. _____

34. Show what should replace BBB. _____

35. Show what should replace CCC. _____

36. Consider the call `(get 'fake-part 'arctic)`. **Draw** a very simple representation of a table that would allow this call to work correctly. Be sure to clearly indicate what type of object is stored in the table.

37. You wish to test the performance of a program/function you wrote (perhaps **add**), and implement and test alternative versions of the function when the whole system is running. What program design characteristic will help ensure that you can do so without having to modify the whole system, and without upsetting system users?

38. (15 pts) What is the purpose of the "package" we saw in chapter two? Does it have an analogous form in the other two approaches to generic ops we examined? Explain.

39. (15 pts) Write message-passing code that allows two "integers" to be added together.

40. (10 pts) Write a high-level **add** procedure for a data-directed programming system that takes a variety of representations as arguments.

41. (12 pts) Write dispatching-on-type code that shows how you could have a generic operator for adjoining a number to a set. The set could be ordered or unordered. Assume you have set type specific operators already written.

42. (15 pts) Evaluate the expressions shown and fill in the diagram. Only use frames provided (you may not need all of them).

```
(define x 7)
(define y 5)
(define f
  (lambda (x) (* x y)))
(define z (f 4))
```

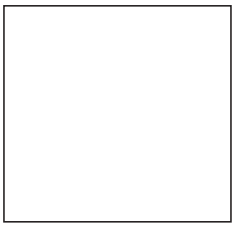
43. (15 pts) Evaluate the expressions shown and fill in the diagram. Only use frames provided (you may not need all of them). Hint: **if** is not a procedure application.

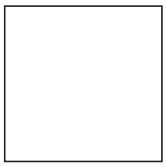
```
(define a 5)
(define (f x y)
  (if (= y 4)
      1
      (f (* x y) (- y 1))))
(define b (f a a))
```

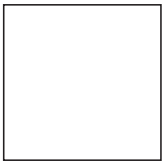
44. (16 pts) Evaluate the expressions shown and fill in the diagram. Only use frames provided (you may not need all of them).

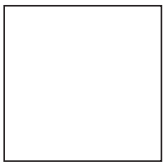
```
(define x 5)
(define (f x)
  (lambda (z)
    (set! x 3)
    (* z x)))
(define z ((f x) x))
```

G

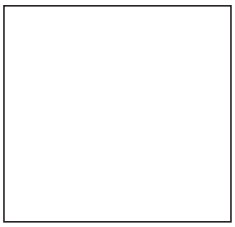


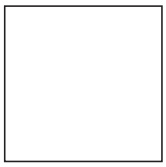


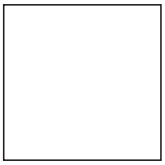


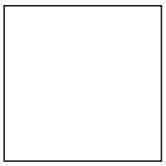


G









G