CISC280 Midterm 1 Sample Questions, Spring 2009

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 10 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. (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)`

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

3. (7 pts) Write the interpreter's representation of this structure:

4. (10 pts)Write the simple recursive-process procedure product-list, which takes two non-nested lists of integers, and returns the list of integers resulting from multiplying each element of blist by the corresponding element in alist. If one list is longer, the remaining elements are multiplied by 1.

```
> (product-list '(4 5 6) '(1 2 7))
(4 10 42)
> (product-list '(1 2 3) '(1 2 3 6))
(1 4 9 6)
>
```

5. (5 pts) Write the simple recursive procedure **scale** that multiplies each member of a list by some factor. Do not use the sequence toolkit.

```
> (scale 2 (list 1 2 3))
(2 4 6)
```

6. (5 pts) Write **scale** using your sequence toolkit.

7. (5 pts) Write a **call** to your sequence toolkit that generates the list of the cubes of the even numbers from 1 to 1000. You may use lambdas, but do not separately define procedures.

8. (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))
```

9. (10 pts) Write a generic procedure **get-numer** that uses dispatching-on-type to return the numerator of a rational number. Rational number representation layers below this procedure include frogs and crystals.

10. (8 pts) Consider a file of functions that use dispatching-on-type to implement a rational number system over multiple representations. Which process would be more error prone, adding a new rational operator (e.g. mult-rat) or adding a new data representation? Explain your answer if you would like credit.

11. (10 pts) Write the procedure **intersection** for an ordered list representation of a set.

Complete the following procedure definitions.

12. (5 pts)

```
(define (fast-expt x y)
  (cond ((= y 0) 1)

        ((odd? y) _____________________________________________

        _____________________________________________________
```

13. (5 pts)

```
;You can put this on more than one line if you wish.
(define (filter proc alist)

  (accumulate _____________________________________________________________________
```

14. (5 pts)

```
(define (equals? a b)
  (cond ((pair? a)

        (else (eq? a b))))
```

15. (5 pts)

```
(define (append a b)
  (define (iter x y)
    (cond ((null? x) y)

          (else _____________________________________________________________
  (iter (reverse a) b))
```

16. (12 pts) Write sort as an accumulation. You may write a helper procedure if needed.

17. (5 pts) What is big O for time for the sorting procedure in 14? Explain.

18. (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))
```

19. (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))
```

20. (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)))
```

21. (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.

> _____

22. (5 pts)In lab 6 we defined a procedure cons. Fill in the three blanks:

```
(define cons

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

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

23. (6 pts) How can you test if a program was designed using proper data abstraction? Be concise.

Trees

24. (5 pts) How many nodes do we expect in a balanced binary tree of height 9? Show your calculations. Show your answer as an integer.

25. (5 pts) How many nodes do we expect in the **bottom row** of a balanced binary tree of height 22? Show your calculations.

26. (5 pts) Assume a tree of 185 nodes is as balanced as it can be. What is the height of the tree? Show your calculations. Show your answer as an integer.

27. (5 pts) Consider the procedure adjoin-tree that adds one element to a binary search tree. What is big O? Explain.

28. (6 pts) The procedures `tree->list` and `list->tree` can be used to improve the performance of union-tree. Explain how.

29. (5 pts) Xenon is proposing using `tree->list` and `list->tree` to improve the performance of adjoin-tree. Explain your reaction to this proposal.

30. (8 pts) Assume you are given procedures **make-tree**, **entry**, **left**, and **right**. Write the procedure **sum-tree**, a tree-recursive procedure which sums all the elements in a tree.

31. (4 pts) Suppose a tree contains K nodes. What is the big O of sum-tree in terms of K? Explain.

32. (4 pts) Would you expect the typical run time to be better than, worse than, or the same as big O predicts? Explain.

33. (3 pts) Extra Credit: The Joel Spolsky article refers to a process that uses two sequence operators, map and accumulate (but he uses another name for accumulate). What was the name of the process and what company uses it?

34. (30 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 5 ())`

    (b) `(cons () ())`

    (c) `(list () ())`

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

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

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

    (g) `(append (list 5 6)(list 3 4))`

    (h) `(apply + (list 1 2)(list 3 4))`

    (i) `(cdr (list 3 4))`

    (j) `(if (equal? (list 1) (list 1)) 2 3)`

    (k) `(quote (list 1 2))`

    (l) `(cons '(1) ())`

    (m) `(lambda (x) (- x x))`

    (n) `(cond (#f 3 4)(#t 5 6)(else 7))`

    (o) `((if (not #t) 7 odd?) 8)`

35. (7 pts) Draw the box and pointer notation for (1 (2 (3 (4)) 5 6))

36. (7 pts) Write the interpreter's representation of this structure:

37. (10 pts)Write the linear recursive procedure **map** which takes a procedure and a list, and returns the list that results from applying the procedure to each element in the argument list.

38. (5 pts) Write the linear recursive procedure **shift** that moves each member of a list along the number line by some distance. Do not use the sequence toolkit.

```
> (shift -2 (list 1 2 3))
(-1 0 1)
```

39. (5 pts) Write the procedure **shift** again using your sequence toolkit.

40. (5 pts) Write a **call** to your sequence toolkit that calculates the number of prime numbers from 1 to 1000. You may use lambdas, but do not separately define procedures. Assume you are given a pre-defined predicate called **prime?**.

41. (5 pts) Write the linear recursive procedure **copy** that generates a list with n copies of an argument:

```
> (copy 'a 3)
(a a a)
```

42. (5 pts) Now write a tail recursive version of **copy**.

43. (8 pts) What is big O for time of linear recursive copy? What about for tail-recursive copy? Explain.

44. (8 pts) What is big O for space of linear recursive copy? What about for tail-recursive copy? Explain.

Complete the following procedure definitions.

45. (5 pts)

```
(define (append a b)

  (cond ((null? a)_____________________________________________

        _____________________________________________

        _____________________________________________
```

46. (5 pts)

```
;You can put this on more than one line if you wish.
(define (filter proc alist)

  (accumulate _____________________________________________________________________
```

47. (8 pts)

```
(define deep-reverse
  (lambda (elts)
    (cond ((null? elts) ())
  ((list? (car elts))
```

_____

_____

_____

48. (5 pts)

```
(define (reverse-iter a b)
    (cond ((null? a) b)
```

_____

_____

_____

49. (9 pts) As discussed in class, what are the three elements required to create a programming language? Name them and give an example of each in Scheme.

50. (10 pts) Xanax is having trouble with her lab results. Using the following procedure, she expects that it will take longer to calculate a result for larger y numbers. But her results show that the program runs faster on input x = 2, y = 24 than it does on input x = 2, y = 15. Show exactly why this is happening in terms of procedure calls to proc. You may assume that procedures f and g do not substantially affect run-time.

```
(define (proc x y)
  (cond ((= y 0 ) 1)
        ((even? y)
         (f (proc x (/ y 2))))
        (else
         (g x (proc x (- y 1))))))

;1000 runs on input 2, 24: 0.07 seconds
;1000 runs on input 2, 15: 0.09 seconds
```