CISC280 Final Exam, Fall 2005

NAME _____

General instructions:

DO NOT WRITE YOUR NAME ON ANY PAGE EXCEPT THIS ONE!

Turn off any noise making device, especially **CELL PHONES**. You may lose up to one letter grade if your device disturbs the peace of the exam.

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

There are seven pages worth a total of 110 points. Read the problems very carefully. Identify what kind of answer the problem requires. 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.

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 the Scheme primitives we use in class.

Scheme programming

1. (8 pts) Write the simple recursive procedure map to behave as follows (you may assume the first argument is a single procedure).

```
> (map (lambda (x) (+ 1 x)) '(1 2 3))
(2 3 4)
```

2. (4 pts) Write the recursive procedure "loop" that takes no arguments and never returns.

Complete the following procedure definitions.

3. (4 pts)

```
;Assume the lists are the same length
;> (interleave '(2 3 4) '(5 6 7)) -> (2 5 3 6 4 7)
(define interleave
  (lambda (alist blist)
    (cond ((null? alist) ())
          (else
```

4. (4 pts)

```
;Assume you have the primitives you need.
;Allows comparison of hexes for storage in a binary tree only.
(define (less-than-hex? h1 h2)
  (cond ((< (get-x-coord h1)(get-x-coord h2)) #t)
        ((> (get-x-coord h1)(get-x-coord h2)) #f)
        (else
```

5. (4 pts)

```
;(accumulate + 0 '(1 2 3)) -> 6
(define (accumulate op init alist)
  (cond ((null? alist) init)
        (else
```

6. (14 pts) Draw the environment diagram as shown in class. Use the frames in the order they are created, starting at upper left.

```
(define (f a b) (* a b x))
(define g (lambda (x)
            (set! x 7)
            (f 1 2)))
(define x 2)
(define z (g 6))
```

7. (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. If you get confused, try drawing a little environment diagram. :)

```
> (define (f x)
     (/ x 2)
     x)
> (define a 20)
> (f a)

a. _____

> (f a)

b. _____

> (define b 8)
> (define (g x)
     (set! x (* x 2))
     x)
> (g b)

c. _____

> (g b)

d. _____

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

e. _____
> (h 100)

f. _____

> (k)

g. _____
```

Generic Operations

8. (4 pts) Write a **single call** to the apply-generic procedure below that will perform the `mult` operation on two numeric arguments, each tagged as type `blt`.

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


>
```

9. (4 pts) Draw a very small, simple table that would enable that call to work. Show ONLY column and row headings that are NECESSARY, as well as contents (as I showed them in class).

10. (3 pts) What two structures did we discuss that can help with the combinatorial explosion of types?

11. (2 pts) What is it called when the interpreter makes use of those structures to help with the multitude of argument types by affecting the types of arguments?

SHORT Answers: For the following questions, focus on coding aspects that are characteristic of the approach mentioned. Read carefully!

12. (3 pts) Describe in general terms what you write/modify to add a new *operation* to a dispatching-on-type system.

13. (3 pts) Describe in general terms what you write/modify to add a new *operation* to a data-directed type system.

14. (3 pts) Describe in general terms the piece or pieces of code you write to add a new *type* to a message-passing type system.

15. (3 pts) In data-directed type systems, how did we avoid procedure name conflicts between procedures without changing the names?

Evaluation

16. (12 pts) In the meta-circular evaluator, when a cond expression is passed to eval it is transformed *before* it is recursively evaluated. Follow the code in the evaluator and show the transformed version of the following cond expression.

```
(cond ((> x 2) (f x))
      ((> x 3) (g x) x)
      (else 7))
```

17. Under one of the models we used to describe evaluation, this code is ambiguous. Which model?

```
(define test
  (lambda (x)
    (set! x (+ 1 x))
    x))
```

(3 pts)_____

18. What property of this model makes the code ambiguous?

(3 pts)_____

19. One procedure in the meta-circular evaluator is responsible for ensuring that this code segment from 17 is not ambiguous (assuming all necessary primitives are defined). What is the name of the procedure?

    (4 pts)_____

Project

20. (5 pts) When bridges were added to the project, was it necessary (as discussed in class) to change how the AI rated moves based on clusters? Explain **briefly** why or why not.

Box and Pointer

21. (4 pts)

```
> (define a (list 1 2 3))
> (set-cdr! (last-pair a) a)
```

    Draw the box and pointer notation for a.

22. (2 pts)

```
> (define b (list 1 2 3))
> (set-cdr! (last-pair b) (car b))
```

    Draw the box and pointer notation for b.