CISC280 Final 1, Spring 2003

NAME _____

General instructions:

There are 10 problems worth a total of 197 points, and an extra credit problem worth 5 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 or irritate the grader, so don't make stuff up.

Do not use assignment unless it is necessary to solve the problem.

You may use any of the simple procedures we use regularly in class, and the following procedures (unless you are asked to define them):
apply
append
reverse
map
accumulate
filter
enumerate
stream-car
stream-cdr
eq?
equal? (only if needed)

1. 10 pts. Let p1 and p2 be procedures that take single numeric arguments. Write a procedure `max-proc` that takes two such procedures and a single number, and returns the procedure that has the highest return value for that input.

2. 10 pts. Write a predicate (subset? alist blist) that returns true if and only if every member of alist is also a member of blist. Elements may be numbers or symbols. Write any supporting procedures you use.

3. 18 pts. Draw the environment model showing the results of evaluating this sequence. Use the enumerated frames, in order. Be sure to write notes, as I do in class, so that I can give partial credit. If a value is changed, simply cross out the previous value once.

```
(define x 7)
(define (f x y)
  (define (g x) (set! x 5))
  (+ x y))
(set! x (f 2 3))
```

4. 18 pts. Draw the environment model showing the results of evaluating this sequence. Use the enumerated frames, in order. Be sure to write notes, as I do in class, so that I can give partial credit. If a value is changed, simply cross out the previous value once.

```
;watch parens!
(define (f x y)
  (define (g) (set! x 5) x)
  (lambda (x) (+ x (g y))))
(define x ((f 2 3) 7))
```

5. 25 pts. Breathe. Then write the procedure **eval** for the meta-circular evaluator. I do not expect your code to be perfect; you are trying to communicate your knowledge of the meta-circular evaluator. Your procedure names don't have to match those of the text, but they should clearly indicate that you understand what is happening in each case. Use comments to clarify if you aren't satisfied with your code. Hint: Do the other problems first, then go through the code you (and I) have written to be sure you have all (or most) of the cases you need.

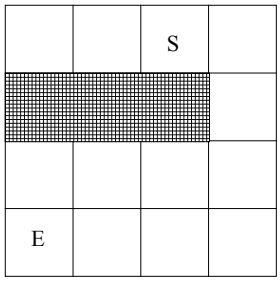
b. 5 pts. What argument does eval need that apply does not? Why?

```
(define (solve-maze maze time)
  (let ((start-time (current-seconds))
        (start-soln-set (MakePSset (MakeNullPartialSolution maze))))
      (define (time-out?) (> (current-seconds) (+ start-time time)))
      (define (solver solution-set)
        (if (or (time-out?) (EmptyPSset? solution-set))
            ()
            (let ((bestPS (getBestPartialSolution solution-set)))
              (if
                (IsSolutionMaze? bestPS maze) (GetSetOfMoves bestPS)
                (solver
                 (CombinePSsets
                  (ExpandPartialSolution bestPS maze)
                  (RemovePartialSolution bestPS solution-set)))))))
  (solver start-soln-set)))

(define (GetBestPartialSolution PSset)
  (define (GBPSiter set bestPS bestRating)
    (if (emptyPSset? set) bestPS
        (let ((rating (GetEval (GetFirstPartialSolution set))))
          (if (<= bestRating rating)
              (GBPSiter (AllButFirstPartialSolution set) bestPS bestRating)
              (GBPSiter (AllButFirstPartialSolution set)
                        (GetFirstPartialSolution set) rating)))))
  (if (emptyPSset? PSset)
      (error "empty set passed to GetBestPartialSolution")
      (GBPSiter PSset (car PSset) (GetEval (car PSset)))))

(define (ExpandPartialSolution ps maze)
  (let ((moves (list MoveLeft MoveRight MoveUp MoveDown)))
    (accumulate addPartialSolution theEmptyPSset
       (map (lambda (path) (makePartialSolution path maze))
            (filter (lambda (x) (not (null? x)))
                    (map (lambda (path) (if (null? path) ()
                                            (NoRetrace? path maze)))
                         (map (lambda (proc)
                                (proc (GetPath ps) maze)) moves)))))))
```

6. 20 pts. For the maze shown, and using the code given, answer the following questions:



a. Exactly how many times will ExpandPartialSolution be called?

b. Refering to each square as a list of row and column, show the order of the squares chosen for expansion. Your answer should look like this: (7 6) (7 5) (4 5) ....

7. a. 18 pts. Show what is returned by the interpreter for each of the following, then describe the result (or explain why it produces an error). If a message is returned (like #proc:f or #error), show the approximate message.

Example:

> (lambda (x) (+ x 1))
result: #proc
description: an unnamed procedure that returns argument + 1

>(define ones (cons 1 (delay ones)))
result:
description:

>(cdr ones)
result:
description:

>(define (ones) (cons 1 (delay (ones))))
result:
description:

>(force (cdr (ones)))
result:
description:

>(define mystery (cons 1 (delay (+ 1 (car mystery)))))
result:
description:

>(stream-cdr mystery)
result:
description:

b. 7 pts. Define a procedure that returns the infinite stream 0, 1, 2, ... You may not use any stream utilities.

c. 8 pts. Write the stream utility (stream-enumerate-interval $< low > < high >$).

8. a. 4 pts. Consider an efficient intersection procedure for ordered list representation of a set. What is the complexity, in big O notation?

   b. 8 pts. How many times would it be called for the two sets $\{1, 2, 3\}$ $\{1, 3, 4\}$, assuming they were represented in the proper way? SHOW EACH of the calls.

9. a. 4 pts. Draw the box and pointer notation for the structure (4 () 5 ((6) 7)).

   b. 4 pts. Define a list of numbers with a cycle in it, so that when it is traversed, the numbers encountered are: 1, 2, 3, 2, 3, 2, 3, ...

c. 15 pts. Write a procedure **cycle?** that detects a cycle in a list without modifying the structure. The cycle may begin anywhere in the list.

d. 3 pts. What is the order of growth of space of the procedure in part c if the list is size n? Use big O notation.

10. a. 10 pts. What are generic procedures? Why do we use them?

    b. 10 pts. What is message-passing?

11. Extra Credit:

    5 pts. Which of the following did Terry say (approximately) in the last class:

    "Recursion is always better than loop structures."

    "Sometimes loops are a better solution than recursion."

    "If Scheme had loops, it would no longer be semantically clean."

    "Most commercial Scheme compilers do have loop structures."

    and what was he referring to when he said it?