CISC280 Final Exam, Fall 2004

NAME _____

General instructions:

There are 18 questions worth a total of 116 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 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
eq?
equal? (only if needed)
set!, set-car!, set-cdr! (only if needed)

1. (6 pts) What are the three methods shown in class to evaluate a sequence of expressions in Scheme?

2. (10 pts) How do objects, such as the message-passing bank accounts we created, differ from streams with respect to time?

The Meta-evaluator

3. (5 pts) How do you start the meta-evaluator?

4. (5 pts) Describe exactly what changes to the meta-evaluator you would make to add a new primitive operation, subtraction.

5. (5 pts) Write the function primitive-procedure-names that belongs in the meta-evaluator (see setup-environment).

6. (10 pts) Write the function list-of-values that belongs in the meta-evaluator (see eval).

Memory

7. (6 pts) Define a fcn last-pair that walks through a proper list and returns the terminating pair.

8. (10 pts) Use last-pair to write `append!`, which appends two lists.

9. (2 pts) What is the order (big O) of `append!` ? Be specific.

10. (2 pts) What is the order (big O) of simple recursive `append`? Be specific.

11. (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`.

12. (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!` .

13. (5 pts) Is there any advantage to using `append` vs. `append!` ? When would one be better than another?

Halt

For the halt proof, we assumed that we were given the working code for an imaginary function `(halt? <program> <input>)` that returns true if a program stops when given the input, and false otherwise. We then wrote

```
(define (try arg)
   (if (halt? arg arg) (spam)
      'cease))
```

14. (5 pts) Write a procedure `spam` that makes the proof work.

15. (5 pts) What argument would you call `try` on to derive a contradiction?

16. (5 pts)What is the significance of the halt proof to computer science?

Your answer may not go below this line.

Hex Game

Suppose a hex is a list `(<letter> <x> <y>)`. You choose a board representation where all hexes that have been played are stored in a single binary search tree. Sort hexes first by x-coord, then by y-coord, i.e. `(X 2 5)` is less than `(X 3 2)`, and `(X 2 5)` is less than `(X 2 6)`.

17. (15 pts) Write a procedure to find if a hex has been played by either player; it takes a hex and a tree as arguments and returns the hex (if found) or null. Write whatever supporting procedures you need. Use appropriate procedural abstraction to get full credit.

18. (10 pts)

Discuss how the performance of this representation compares to storing the played hexes in a simple list. Be specific about the complexity and interaction between the game play and the board representation.

End of Exam. Total Points: 116