

Sample Final Exam

Problem 1 (10 points): List Mutation

Suppose a Scheme interpreter evaluates the following expressions:

```
(define (append lst1 lst2)
  (if (null? lst1)
      lst2
      (cons (car lst1) (append (cdr lst1) lst2))))

(define a '(a b c))
(define b (cons a a))
(define c (append a a))
```

Draw the box and pointer representation of the data structures a, b, and c.

What would the Scheme interpreter print as the value of b?

The Scheme interpreter now evaluates the following expressions:

```
(set-cdr! (cdr b) nil)
(set-cdr! (cdr c) a)
```

Draw the box and pointer representation of the data structures a, b, and c after these expressions are interpreted.

What would the Scheme interpreter now print as the value of b?

Problem 2 (20 points): Streams

Consider the following definitions for `add-streams` and `mul-streams`.

```
(define (add-streams s1 s2)
  (cons-stream (+ (stream-car s1) (stream-car s2))
               (add-streams (stream-cdr s1) (stream-cdr s2))))

(define (mul-streams s1 s2)
  (cons-stream (* (stream-car s1) (stream-car s2))
               (mul-streams (stream-cdr s1) (stream-cdr s2))))
```

Capture the pattern in the definitions above by writing `map-two-streams`, which takes a function and two streams as parameters.

Use `map-two-streams` to define `min-two-streams`, which takes two streams and returns a stream containing the minimum values of the corresponding elements of the stream.

For example,

```
(min-two-streams integers twos)
```

would return

```
1 2 2 2 2 2 2 2 ...
```

Problem 2 continued:

The following stream definitions are used to define the stream of factorials.

```
(define ones
  (cons-stream 1 ones))

(define integers
  (cons-stream 1 (add-streams ones integers)))

(define fact
  (cons-stream 1 (mul-streams fact integers)))
```

How many multiplications are performed when computing the n^{th} factorial value in the `fact` stream? Explain your answer.

If there were no memoization in Scheme, how many multiplications would be performed when computing the n^{th} factorial value in the `fact` stream? Explain your answer.

Problem 3 (15 points): Object Oriented Scheme

Refer to the game.scm handout for this problem.

Let's create a new type of person in our system, called an `opl-lecturer`. The `opl-lecturer` should be able to `bring-cookies-to-exam`. Additionally, the `opl-lecturer` should say "Scheme is fun" after anything that she is asked to say. Fill in the blanks in the following procedure. Assume that `cookie-store` is a magical place that will continually manufacture `cookies`.

```
(define (make-opl-lecturer name birthplace threshold)
  (let (<exp1>)
    (lambda (message)
      (cond (<exp2>
            (lambda (self)
              (let ((location (ask self 'place)))
                (ask self 'move-to cookie-store)
                (ask self 'take cookies)
                (ask self 'move-to location))))
            ((eq? message 'say)
             <exp3>)
            (else <exp4>))))))
```

Write your answers here.

<exp1>

<exp2>

<exp3>

<exp4>

Problem 4 (15 points): Metacircular Evaluator – Changing the syntax of our language

After spending a semester writing prefix expressions, Louis Reasoner decides that he would like to change the language of our metacircular interpreter to accept infix expressions for `uml:+`, `uml:-`, `uml:*`, and `uml:/`, where there are only two arguments. For example, instead of writing `(uml:+ x 2)`, we would now write `(x uml:+ 2)`. There will be no change to how we call other procedures.

Refer to the `mc-eval.scm` file for writing your solution. If you are adding code to the system, please write it below. If it is a line of code to be inserted into an existing procedure, be specific and clear about where it should be inserted. If you are modifying existing procedures, it would probably be clearer if you were to rewrite the procedure here.

Problem 5 (15 points): Metacircular Evaluator – Changing the way we store variables and values in an environment frame

In the metacircular evaluator, variables and values are simply `consed` together in an environment frame, using `make-frame`:

```
(define (make-frame variables values)
  (cons variables values))
```

Rewrite `make-frame` so that the variables and values are paired in `cons` cells, with each `cons` cell the element of a list. Recall that `extend-environment` checks to see if there are the same number of variables and values, so you do not need to do this in `make-frame`.

For example, evaluating

```
(make-frame '(a b c) '(1 2 3))
```

would return the list

```
((a . 1) (b . 2) (c . 3))
```

Problem 6 (15 points): Garbage Collection

Show the results of running mark-and-sweep garbage collection on the following memory. The root is P2.

0	1	2	3	4	5	6	7	8	9
P2	P7	P5	P6	N2	N1	N2	P2	N6	P5
P7	N7	N3	P1	P1	N4	P7	P3	N7	P4
0	0	0	0	0	0	0	0	0	0

Root:

Free:

Show the results of running stop-and-copy garbage collection on the following memory. The root is P2.

0	1	2	3	4	5	6	7	8	9
P2	P7	P5	P6	N2	N1	N2	P2	N6	P5
P7	N7	N3	P1	P1	N4	P7	P3	N7	P4

10	11	12	13	14	15	16	17	18	19

Root:

Free:

Problem 6 continued:

When would running stop-and-copy garbage collection be better than mark-and-sweep?

What is the primary advantage of mark-and-sweep over stop-and-copy?

Problem 7 (10 points): The Analyze Evaluator

Why would we want to use the analyze evaluator instead of the metacircular evaluator?
Explain in one or two sentences.

Are there any drawbacks to the analyze evaluator? Explain your answer.

Extra Credit Problem (10 points):

How would you need to change `lookup-variable-value` to work with our new method of storing variables and values from Problem 5? Write the code here.

Is there anything else in the metacircular interpreter that would need to be changed for our new method for storing variables and values in frames? You do not need to write the code if any changes are necessary. Just state what needs to be done in existing procedure(s), if applicable.