Computer Science 61A - Spring 1996 - Harvey - Final Exam

Answers to CS61 A Final of May 23, 1997
We repeat the questions; some have typos corrected with respect
to the actual exam handed out.

Question 1 (5 points):
Let us represent a set S of unique expressions, atoms or
pairs, as a Lisp list of those expressions.
As you know, H is a subset of S if each element e of H is also an
element of S.
The powerset of S is the set of all subsets of S.
The powerset of S includes the empty set and S.

Here is a recursive way of thinking about powersets as lists:
First, the powerset of () is (()). That is, it has one element, namely ().
Next, assume that S has at least one element e, and
let K be the rest of S. Let P be the powerset of K.
The powerset of S is the union of P and a set formed
by sticking e in front of each element of P.

Here are two examples:
The powerset of (1 2 3) is (() (1) (2) (3) (1 2) (1 3) (2 3) (1 2 3))
The powerset of (1 (2 3)) is (() (1) ((2 3)) (1 (2 3)))

What is the powerset of '(1 (2 3) 4)?

Answer: 1 point. no partial credit

(() (1) ((2 3)) (1 (2 3)) (4) (1 4) ((2 3) 4) (1 (2 3) 4))

all 8 elements.

What is the length of the list representing
the powerset of '(1 2 3 4 5)?
Answer: 1 point. no partial credit. 32. (2^5 = 32)

Write a procedure powerset which takes as its only argument a set S and
returns the powerset of S.

Answer: 3 points total

```
(define (powerset set)
  (if (null? set)
      (list '())   ; 1 point for the base case
    (append (powerset (cdr set))  ; 1 point for the append
            (map (lambda (s) (cons (car set) s))  ;1 point for the map/lambda
                 (powerset (cdr set))))))
```

Some people copied the "subsets " program. That was ok except if
you called it subsets instead of powerset.

Question 2 (7 points):
Draw the environment diagram resulting from the
following inputs to the Scheme system. What is
the value of the global variable x at the end?

```
(define x 10)

(define (foo x) (define (bar y) (set! x y) x)   bar)

((foo 50) 60)
```

♀

Answer: global env has x:10 , foo pointing to function def. 1pt

sub environment has x:50, then 60 and bar in it. 1 pt
sub sub environment has y:50 in it 1 pt
correct arrows 2 pt

x is 10 at the end 2 pt

Question 3 (5 points):
Consider the following piece of code:

```
(define foo (lambda (x) (- y x)))
(define bar foo)
(define y 7)

(parallel-execute
 (lambda () (set! foo (let ((z y))
                        (lambda (x) (+ (bar x) z)))))
 (lambda () (set! y (foo 3)))
)
```

What are the possible values of y?

ANSWER: 4, 11 (2 pts.) 1 off if you had wrong answers too.

The easy part is if branch 2 runs first.
y is set to (- 7 3), or 4

Next foo is redefined as though we did
```
  (define foo (let ((z y))
                        (lambda (x) (+ (bar x) z))))
```
which is actually the same as
```
     (define (foo x)(+ (bar x) 4))
```
where we have just put the value of z, namely the current
value of y, namely 4 in there. So foo is
```
    (define (foo x)(+ (- y x) 4))      (*)
```

in which case (foo 3) will be (+ (- 4 3) 4) or 5

Another possibility is that branch 1 is executed first,
redefining foo as though it were (define (foo x)(+ (bar x)) 7)

where we have just put the value of z, namely the current
value of y, namely 7 in there. So foo is
really the same as (define (foo x)(+ (- y x) 7))   (**)

Then running branch2 sets y to (foo 3) which is (+ (- 7 3) 7) or 11.

A third possibility is that one starts to run branch 1 and after
binding z to y, namely 7, branch 2 is run changing the global
y to 4.
Then foo is defined as the equivalent of (define (foo x)(+(- y x) 7)
with global y being 4. (***)

What are the possible results of computing (foo 3) after all the
values have settled down?

Ans;
 with foo defined as in (*) and y set to 4, (foo 3) is 5
 with foo defined as in (**) and y set to 11 (foo 3) is 15
 with foo defined as in (***) and y set to 4 (foo 3) is 8
grading: this was tough. We gave you 3 points if you got
at least 2 of these, and no wrong answers. If you got some wrong

ones we took off 1

Question 4 (8 points):
In this age of health-consciousness, the adventure game project needs some revision.
We are going to expand the food class and also allow a separate class for fat-free foods. For reference, here is the food class from the online solutions:

```
(define-class (food name calories)
        (parent (thing name))
        (method (edible?) #t))
```

Your job is to modify this class so that one can type:

(instantiate food 'bagel 240 1.333)
to create a food object instantiated with the name bagel, a calorie value of 240, and a fat content of 1.333 grams.
Additionally, the user should be able to type:
(instantiate fatfreefood 'carrot 50)
to create a food object (presumably a standard serving of carrot) with the name carrot and a calorie value of 50 (the fat content of 0 will be understood).

If obj is a food or fatfreefood object then it should respond as follows:

(ask obj 'edible?) should return #t.

(ask obj 'name) should return the name of the object.

(ask obj 'fat-in-grams) should return obj's fat in grams.

(ask obj 'pct-calories-from-fat) should return obj's percent of calories from fat. Each gram of fat contains 9 calories. Therefore our bagel has about 12 calories from fat, and is 12/240 or 5 percent calories from fat.

(ask obj 'calories) should return obj's calorie value.

(ask obj 'what-are-you) should display a message "I am food." for regular food objects; for fatfreefood objects, it should display whatever message is displayed for regular food objects, plus this message: "I am fat-free." Keep in mind that if we later change the food-description method for regular food objects, the result of asking a fatfreefood what-are-you must change as well.

Correct solutions should demonstrate understanding of OOP and inheritance. If your solution is significantly longer than needed it will not get full credit.
By the way, a recommended diet has less than 30 percent of calories from fat. Most Americans have an unhealthy 40 percent or more ratio.

ANSWER: 5, 3 points for the classes.

```
(define-class (food name calories fat-in-grams) ; 1
```

```
        (parent (thing name))                    ; 1
        (method (edible?) #t)                    ; 1
        (method (pct-calories-from-fat) ; 1
               (* 100 (/ (* fat-in-grams 9) calories))) ;ok if not * 100.0
        (method (what-are-you)                   ; 1
                (display "I am food.")))


(define-class (fatfreefood name calories)
        (parent (food name calories 0))          ;1
        (method (what-are-you)
                (usual 'what-are-you)            ;1
                (display "I am fat-free")))      ;1
```

Common PITFALLS

(1) People defining a redundant calories method. lose a point
(2) People defining a redundant name method. ok, since we had a typo
of "name?" for name.
(3) People failing to use "usual". lose a point
(4) People defining unnecessary methods or class-vars, lose a point
    to handle the fat-content of fatfreefood objects.
(5) People not making fatfreefood inherit from food.
(6) People returning strings instead of displaying them. lose a pt.

Not using inheritance: lose 2 points at least.

Question 5 (5 points):
This question concerns the Logo interpreter in Project 4.
At some points the interpreter detects erroneous constructions
and returns to the top-level loop of the
Scheme interpreter. We ask you to make a change so that it
returns to the top-level of the Logo interpreter.

The logo interpreter behaves as follows:

```
> (initialize-logo)
? to foo :x
-> print 5
-> :x
-> print 6
-> end
? foo 10
5
ERROR: You don't say what to do with 10
>
```

Note the `` >'' prompt which means you are
returning to the Scheme top level.
Now modify the above definition of eval-sequence
 so that the logo interpreter would behave the same as above except

```
 .... same as above ..
? foo 10
5
You don't say what to do with 10
?
```

In particular, note the ``?'' prompt which means you are
returning to the Logo top level. Show your changes by modifying
 eval-sequence or adding code below.

```
(define (eval-sequence exps env)
  (if (null? exps)
      '=no-value=
      (let ((value (eval-line (make-line-obj (car exps)) env)))
        (cond ((eq? value '=stop=) '=no-value=)
              ((and (pair? value) (eq? (car value) '=output=))
               (cdr value))
              ((not (eq? value '=no-value=))
               (error "You don't say what to do with" value))
              (else (eval-sequence (cdr exps) env))))))
```

ANSWER: 5 pts total:

There were lots of ways of doing this. All involved

```
replacing  ;1 pt (where to change)

(error "You don't say what to do with" value)
```

with something.

Some people figured out that all you had to do was put in

```
value
```

and the error would propagate to the top level, giving the right behavior. This is kind of fragile. It works for this example but is not a good general solution. We expected something like

```
(display "You don't say what to do with ")  ;1 pt
 (logo-show value) ; use the abstraction!    ;2 pt
 (driver-loop)  ;1 pt
```

and took points off if you just display'ed value... logo-show or logo-print is better.. initializing logo is NOT right. you need to re-enter the driver-loop.

Another solution we allowed used logo-print to give a message and then return =no-value=

In fact we gave full or nearly full credit for most solutions that had glitches but showed a basic understanding.

Question 6 (10 points):

(6.a) Which (if any) of the following expressions provided to the meta-circular evaluator gives the message ``ERROR: Unknown expression type''?

(A) #t
(B) '(a ())
(C) 0.67
(D) (define x ())
(E) (set! y 'a)
(F) true
(G) none of the above

ANSWER: (d) ;2 pts. () is unknown type
``true'' is actually known. if you say d and f you got credit.

(6.b) Which (if any) of the same expressions as the previous question given

to the meta-circular evaluator gives the message ``ERROR: Unbound variable''?

ANSWER: E. y is unbound 2 pts

(6.c) How does the meta-circular evaluator know what to do when it encounters the symbol cons in (cons 1 2)?
(Circle all that are correct).

(A) cons is defined as a lambda
(B) cons is defined in the mceval initial environment
(C) cons is a scheme primitive
(D) cons is a special form
(E) none of the above

ANSWER: C 2 pts

(6.d)
In the Amb evaluator, what are the possible expressions that may be produced from

(amb 1 (amb 'a (cons 1 2 3) 'd) 4)

Note that cons takes only two arguments and therefore its use above causes an error.

ANSWER: 1, a 2 pts.

(6.e) Stripping away layers of abstraction, we could rewrite a compatible version of the meta-circular evaluator as (define (mc-eval exp env) (cond ... ((pair? exp) (mc-apply (mc-eval (car exp) env) (map <B> (cdr exp)))) ...)) What is the simplest expression you can write for <B>? (hint: under 30 chars.)

ANSWER: (lambda(r)(mc-eval r env)) ; 2 pts. 1 pt if partly right

Question 7 (3 points):
If you load the file defining the meta-circular evaluator and type (mce) twice, you get this result. Explain the ERROR message in one sentence.

```
(mce)
;;; M-Eval input:
(mce)
ERROR: Unbound variable mce
>
```

Answer: 3 pts. partial credit possible.

mce is not defined as a function known to the meta-circular evaluation since it is not a scheme primitive or a special form.
(mce is defined in SCHEME by loading in mceval.scm)

Question 8 (5 points):
Define, using our query system, a palindrome detecting program. A palindrome is a list whose symbols are arranged to read the same forward as backward. Examples are (10 20 20 10) and (hello world hello).
We do not mean that individual letter order should be reversed like the English palindrome ``A man a plan a canal panama'' just the symbols taken as units.
You may refer to any rules or standard definitions from

the book or your notes. Hint: This is not a hard problem.

ANSWER: (assert! (rule (palin ?x) (reverse ?x ?x))) ; 5 pts
4 points if basically right idea but bad implementation, most
based on based on stripping off first and last letters.
It is certainly possible to do this correctly, by asserting that
that () is a palindrome (?x) is a palindrome,
and that ?z is a palindrome if ?z is the result of appending
  (?r) ?y (?r) and ?y is a palindrome.

2 or 3 points if some idea

usually, 0 points if you wrote a lisp program or used lispval

 Question 9 (3 points):
The corresponding logo-apply and mc-apply functions
are very similar. They differ in their arguments, however.


```
(define (mc-apply procedure arguments) ...)
(define (logo-apply procedure arguments env) ...)
```

Explain, in one sentence, why these differ.

logo-apply implements dynamic scope but mc-apply implements
static scope. (;3 points)
You might add that in dynamic scope the environment is passed
along with the procedure and its arguments; in static scope, the
environment is part of the procedure.

 Question 10 (6 points):

In the homework on the MCEVAL,
we offered a solution using the derived-expression technique for expanding an
 or into one or more if expressions.
This is done by adding a clause in the big EVAL cond


```
((or? exp) (eval (or->if (cdr exp)) env))
```


and then defining


```
(define (or->if exps)
  (if (null? exps)
      #f
      (make-if (car exps)
               (car exps)
               (or->if (cdr exps)))))
```


This is very neat but is unfortunately slightly incorrect.
This would be illustrated by
evaluating the example (or #f (begin (print 'hi) #t)).
What happens and why?
ANSWER
  Assuming print has the side effect of printing, then HI is printed
  twice, not once. 2 pts

(b) Redefine or->if so that it works.

```
ANSWER something like:
  (define (or->if exps)
  (if (null? exps)
      #f
 ;; must evaluate (car exps) only one time
   (list (list 'lambda '(x) (make-if x x (or->if (cdr exps)))) (car exps))) )
;4 pts if right or nearly right

;3 pts if obviously not right but close, e.g.
  e.g. if you used (make-if (car exps) #t (or->if (cdr exps)))
which has wrong semantics, but close.

;2 pts some idea
;0 pts  no idea


 Question 11 (3 points):


(define (union seq1 seq2)
  (cond ((null? seq1) seq2)
        ((not(member (car seq1)  seq2))
         (union (cdr seq1)(cons (car seq1) seq2)))
        (else (union (cdr seq1) seq2))))


The worst-case running time of this procedure,
as a function of  $n$, the length of the longer of  seq1 and
 seq2 is

(A) $Theta(1)$
(B) $Theta(log n)$
(C) $Theta(n)$
(D) $Theta(n^2)$

ANSWER D:
; in spite of typos in the actual test, the answer was the same.

The  worst case would be when the sequences are of equal
length n, in which case
(member e seq2)  is theta(n) where n
and this is done n times.
Hence theta(n^2).   3 pts.  no partial credit
```