CS 61A Midterm #2 — March 11, 2009

Your name _____

login: cs61a–_____

Discussion section number _____

TA's name _____

This exam is worth 40 points, or about 13% of your total course grade. The exam contains six substantive questions, plus the following:

Question 0 (1 point): Fill out this front page correctly and put your name and login correctly at the top of each of the following pages.

This booklet contains eight numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper. This is an open book exam.

When writing procedures, don't put in error checks. Assume that you will be given arguments of the correct type.

Our expectation is that many of you will not complete one or two of these questions. If you find one question especially difficult, leave it for later; start with the ones you find easier.

If you want to use procedures defined in the book or reader as part of your solution to a programming problem, you must cite the page number on which it is defined so we know what you think it does.

READ AND SIGN THIS:

I certify that my answers to this exam are all my own work, and that I have not discussed the exam questions or answers with anyone prior to taking this exam.

If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

_____

0	/1
1	/4
2	/5
3	/6
4	/8
5	/8
6	/8
total	/40

Question 1 (4 points):

What will Scheme print in response to the following expression? If the expression produces an error message, you may just write "error"; you don't have to provide the exact text of the message. **Also, draw a box and pointer diagram for the value produced by the expression.** Don't forget the start arrow!

```
> (let ((x (list 2 3))
        (y (list 4 5)))
    (let ((z (append x y)))
      (cons y z)))
```

Draw your box and pointer diagram here:

Your name _____ login cs61a–_____

Question 2 (5 points):

For each of the following questions, we will present you with an object class, and an item corresponding to that class.

For each item, tell us which of the following *best* characterizes the relationship with the object class:

(a) instance variable
(b) class variable
(c) child class
(d) method
(e) none of the above

class: house, item: chimney

class: automobile, item: drive

class: book, item: comic book

class: animal, item: binoculars

class: computer, item: keyboard

Question 3 (6 points):

Consider the following procedure:

```
(define (twice f)
  (lambda (x) (f (f x))))
```

We'd like to write `twice` as a (primitive) procedure in the Scheme-1 evaluator. (It has to be a primitive because in Scheme-1 only primitive procedures have names!)

For reference, here are the important parts of Scheme-1:

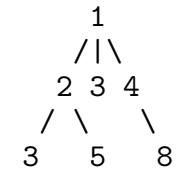
```
(define (eval-1 exp)
  (cond ((constant? exp) exp)
        ((symbol? exp) (eval exp))        ; use underlying Scheme's EVAL
        ((quote-exp? exp) (cadr exp))
        ((if-exp? exp)
         (if (eval-1 (cadr exp))
             (eval-1 (caddr exp))
             (eval-1 (cadddr exp))))
        ((lambda-exp? exp) exp)
        ((pair? exp) (apply-1 (eval-1 (car exp))      ; eval the operator
                              (map eval-1 (cdr exp))))
        (else (error "bad expr: " exp))))

(define (apply-1 proc args)
  (cond ((procedure? proc)        ; use underlying Scheme's APPLY
         (apply proc args))
        ((lambda-exp? proc)
         (eval-1 (substitute (caddr proc)   ; the body
                             (cadr proc)    ; the formal parameters
                             args           ; the actual arguments
                             '()))          ; bound-vars
        (else (error "bad proc: " proc))))
```

Make any changes necessary to implement the `twice` primitive.

Your name _____ login cs61a–_____

Question 4 (8 points):

Write a function `max-sum-children` that takes a Tree filled with *positive* numbers as its argument. It should return the largest sum of the data of the immediate children (not grandchildren) of a node. For example, given the Tree



it should return 9 $(2 + 3 + 4)$, which is the largest sum of data immediately underneath a single node. But if the 5 in the Tree were replaced with a 7, then `max-sum-children` would return 10 $(3 + 7)$.

Question 5 (8 points):

(a) You may assume that the following functions are already defined:
`(attach-type type contents)` –returns a typed datum
`(type x)` –returns the type of a typed datum
`(contents x)` –returns the contents of a typed datum

Suppose you are Imelda Marcos. You have to keep track of your monetary assets and your clothing accessories. Thus possessions come in two types, to which we will attach the manifest types `money` and `clothes`. **Write appropriate** *constructors* **and** *type predicates* **for objects of these two types.**

(b) From time to time you need to tot up these possessions. Because you have assets in the U.S., Japan etc, the values can be 10,000 dollars, 20,000,000 yen or whatever. Similarly, accessories can be stacks of 25 hats or 200 shoes or whatever. These will be represented by the manifest types `dollars`, `yen`, `hats`, `shoes` and so on; you can assume the constructors (`make-dollars` etc.) and type predicates (`dollars?` etc) for these are already written.

We will store in a table the exchange rates between various currencies. For example, to say there are 99.3 yen in a dollar we write:

```
(put 'dollars 'yen 99.3)
```

`Write a conversion function (convert amount from to)`. For example,
`(convert 198.6 'yen 'dollars)` should return 2, and
`(convert 2 'dollars 'yen)` should return 198.6.

This question continues on the next page.

Your name _____ login cs61a–_____

Question 5 continued:

(c) Now **write a function `(+possession x1 x2)` that adds together two possessions and returns a new possession**. Any two monetary possessions can be added once they are converted to the same currency, but hats can only be added to hats, not to shoes. Adding shoes to hats gives an error.

(d) Assuming that complex numbers are represented using manifest types in polar or rectangular form, as in the book, write a generic function `=c` that returns `#t` if its two arguments represent the same complex number. You may use any of the functions `real-part-rectangular`, `real-part-polar`, `real-part`, `imag-part-rectangular`, `imag-part-polar`, and `imag-part`.

Question 6 (8 points):

Write a procedure `depth-tag` that takes a deep list of words as input and outputs the same list except that all words are prefixed with their depth.

For example:

```
> (depth-tag '((never gonna) (give (you) up) !))
((2never 2gonna) (2give (3you) 2up) 1!)

> (depth-tag '(((musketeers))))
(((3musketeers)))
```