Question 1 (3 points):
What will the Scheme interpreter print in response to each of the following expressions? If an expression produces an error message, you may just say "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just say "procedure"; you don't have to show the form in which Scheme prints procedures. Also, draw a box and pointer diagram of the value produced by each expression.

(list (cons 2 3) 4)

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

(cdadar '(e (f) g) h))

Question 2 (1 points):
Fill in the blank in this expression:
_______________ '(G (H I) J)
so that the value of the expression is the letter I.

Question 3 (5 points):
This question concerns the picture project. The \textit{midpoint} of a segment is the point halfway between its two ends. The midpoint of a frame is the point in that frame corresponding to the point (0.5, 0.5) in the unit square:

We want a procedure named \texttt{midpoint} that takes either a segment or a frame as argument, and returns the vector from the origin to the midpoint of the argument. We'll accomplish this in stages.

(a) Write new versions of the constructor \texttt{make-segment} and the selectors \texttt{start-segment} and \texttt{end-segment} that includes the word \texttt{segment} as a type tag in the internal representation.

This question continues on the next page!

Question 3 continued:
(b) We'll assume that the constructor and selectors for frames have been modified similarly so that the word \texttt{frame} is included as a type tag in frames; you need not write these procedures. Now write the procedure \texttt{midpoint} that takes either a segment or a frame as its argument, and returns the vector from the origin to the midpoint of the argument. \textbf{Respect all relevant data abstractions.} If the argument is neither a segment nor a frame, your procedure should return \texttt{#f}.

Question 4 (5 points):

This question concerns message passing. We are going to implement a database of students' grades for various courses. Each course has its own set of assignments. Suppose the assignments for CS 61A have these names:

\textbf{MT1 MT2 MT3 PROJ1 PROJ2 PROJ3 PROJ4}

(Of course the actual list is somewhat longer.) We'd like to be able to say

\begin{verbatim}
> (define dot (make-61a-student '(18 14 16 10 10 8 10))
DOT
> (dot 'MT2)
14
\end{verbatim}

and similarly for the other named scores.

You are \textbf{not} going to write \texttt{make-61a-student}. Instead, write a general \texttt{make-course} procedure that takes \textit{any}
list of assignment names as its argument and returns a student-making procedure for that course. So we'd say

> (define make-61a-student
  (make-course '(MT1 MT2 MT3 PROJ1 PROJ2 PROJ3 PROJ4)))

before using make-61a-student as shown above. You may use the following helper procedure:

(define (lookup name names values)
  (cond ((null? names) #f)
        ((eq? name (car names)) (car values))
        (else (lookup name (cdr names) (cdr values)))))

---

**Question 5 (5 points):**

This question concerns trees, using the constructor make-tree and the selectors datum and children as discussed in lecture. Suppose we are dealing with trees in which the datum at every node is a number. Write a procedure maxpath that takes such a tree as its argument, and returns the largest possible sum of numbers along a path from the root node to some leaf node. For example, in the tree

the largest such sum is 5+2+9, so the procedure should return 16.

**Respect the data abstraction!**

You may use, without defining it, the procedure biggest that takes a nonempty list of numbers as its argument and returns the largest number in the list:

> (biggest '(10 23 7 5))
23