QUESTION 1.
At Depth University, a student must complete at least one advanced class to graduate. However, every advanced class has a prerequisite, which may itself have a prerequisite, and so on. Write a procedure `fast-grad` that, given a prerequisite tree, with constructor `make-tree` and selectors `datum` and `children`, returns the shortest possible list of courses needed to graduate. If there is a tie, `fast-grad` may return any of the shortest lists. You may assume that all leaf nodes are advanced classes, and vice versa.

For example, `fast-grad` called on the following tree can return (CS61A CS70 CS170) or (CS61A CS61B CS184), either one is correct.

```
CS61A
 /   \
CS61B  CS70
 / \
CS164 CS150
```
QUESTION 2.
We want to determine if there is a path from one building to another on campus. Connections are represented like this:

(connected Soda Cory)
(connected Cory Soda) ;; two-way connection

(connected Cory Evans)
(connected Evans Cory) ;; two-way connection

Write rules for \textit{is-connected} that takes a current location and an ending location and a list of future locations and checks if the list is a valid sequence of steps to get to the end location according to \textit{connected}. (Since the final element is a list of FUTURE steps, the current location is NOT INCLUDED in the list, but the ending location matches the last element of the list.)

QUESTION 3.
What are the first 5 elements of the following stream?

\begin{verbatim}
(define powers (cons-stream 1 (stream-map *
    (stream-map +
      ones
      ones
      ones
      ones
      ones)
      powers)))
\end{verbatim}
QUESTION 4.
We want to write a system to calculate the average grades of students. However, we'd like to be able to break them down by type of student: graduate student, undergraduate, lazy, motivated, and so on. A grade is a single number, and it is tagged with the type of student it is associated with:

(define lazy-grade (attach-tag 'lazy 60))

a) Write a procedure populate-records that takes a list of type-tagged student grades and populates a table using the get and put methods discussed in class. It should populate the table with a score total per type of student, and a number of appearances per type of student. For example:

> (define lazy1 (attach-tag 'lazy 60))
lazy . 60
> (define lazy2 (attach-tag 'lazy 70))
lazy . 70
> (define normal1 (attach-tag 'normal 85))
(normal1 . 85)
> (populate-records (list lazy1 lazy2 normal1))
okay ;; return value unimportant
> (get 'lazy 'total)
130 ;; 60 + 70
> (get 'lazy 'appearances)
2 ;; two lazy students

b) Using the total and appearances data, write a procedure get-average that takes in a type of student and returns their average performance. Continuing the example from above, (get-average 'lazy) would return 65. If the type of student did not appear, return 0.

QUESTION 5.
Fill in the blank in the following interaction with the metacircular evaluator:

;;; M-Eval input:
if

______________________________
QUESTION 6.
For each evaluator, will the following expression return a value or cause an error? Circle VALUE or ERROR for each.
> (let ((a 3)
  (b a))
  (+ a 4))
VALUE ERROR The MCE
VALUE ERROR Analyzing evaluator
VALUE ERROR Lazy evaluator

For each evaluator, will the following expression return a value or cause an error? Circle VALUE or ERROR for each.
> (let ((a 3)
  (b a))
  (+ a b)) ;; this line different from the one above
VALUE ERROR The MCE
VALUE ERROR Analyzing evaluator
VALUE ERROR Lazy evaluator

QUESTION 7.
Here is a transcript of a Scheme session. Fill in the blanks

> a
(1 2 (3 4 5) 6)
> b
(1 2 3 4 5)
> c
(1 2 (3 4 5) 6)
> (eq? (cddr b) (caddr a))
#T
> (eq? (caddr c) (caddr a))
#F
> (eq? (cdaddr c) (cdddr b))
#T
> (set-car! (caddr a) 7)
okay
> (set-car! (cdaddr a) 8)
okay
> b

_________________________

> c

_________________________
QUESTION 8.
Draw this baby, but stop drawing when you have a feel for how things will work out in the end.

```
(define x 4)

(define (define-x x) (define x x) x)

(define (confusing x y)
  (let ((z (x y)))
    (set! confusing z))
  (set! x 10)
  (confusing))

(confusing define-x (let ((total 0))
  (lambda ()
    (set! total (+ total x))
    (confusing)))))
```

QUESTION 9.
In the lazy evaluator you are given:
```
(define (actual-value exp env)
  (force-it (mc-eval exp env)))

(define (force-it obj)
  (if (thunk? obj)
    (actual-value (thunk-exp obj)
      (thunk-env obj))
    obj))
```

Consider this modification in all capitals:
```
(define (force-it obj)
  (if (thunk? obj)
    (MC-EVAL (thunk-exp obj)
      (thunk-env obj))
    obj))
```

Assume the following functions are typed at the lazy prompt:
```
(define (identity x) x)

(define (foo a b)
  (+ (bar a) b))

(define (bar a) (* a a))
```

What is the value returned by the lazy evaluator for each of the following expressions? If something abnormal happens, please describe.

(a) (bar 3)

(b) ((lambda (x) (* x x)) ((lambda (x) x) 5))
(c) (foo 6 7)

(d) (let ((a +) (b 4))
              (+ b b))

(e) (identity (identity 6))

**QUESTION 10.**
Write a procedure `useless-square` that performs the squaring function correctly the first time you call it, and thereafter only returns what it returned the first time. For example,

```
> (useless-square 5)
25
> (useless-square 10)
25
> (useless-square 3)
25
```

**QUESTION 11.**
Write rules for the following query that flattens a list:

```
> (flatten (a (b c) d ((e))) ?what)
(flatten (a (b c) d ((e))) (a b c d e))
```