UNIT 2 FINAL

1. (4 points) Communize all the Trees!

Write a procedure \texttt{communize} that takes in any tree as its only argument and returns a tree of depth 1 with the same nodes as the original tree, in some order.

Here is the original Tree. Let’s call it \( t \)

\[
\begin{array}{c}
\text{A} \\
\text{B} & \text{C} \\
\text{D} & \text{E} & \text{G} \\
\text{H} & \text{I} & \text{K} \\
\end{array}
\]

This is the tree returned by \((\text{communize } t)\)

\[
\begin{array}{c}
\text{A} \\
\text{B} & \text{D} & \text{H} & \text{I} & \text{E} & \text{K} & \text{G} & \text{G} \\
\end{array}
\]

Do not make any assumptions about how Trees are implemented. In other words, respect the data abstraction!

Hint: It’s \textbf{much} easier to do this with recursion than it is with higher order functions.
2. (4 points) Assembly Products

We have an ADT for a product. It has 4 parts:

- name, which is a word
- instruction, which is a procedure that takes in a list of integers
- target, which is an integer
- serial, which is a list of 6 integers

We have a constructor, defined as follows:

```
(define (make-product name instruction target serial-num)
  (cons name (list instruction target serial-num)))
```

(a) (2 pt) Write the selectors for each part of the product (name, instruction, target, and serial):

```
STk> (define p1 (make-product 'car car 2 '(2 0 1 -1 8 -3))
STk> (name p1)
car
STk> (instruction p1)
#[subr car]
STk> (serial p1)
'(2 0 1 -1 8 -3)
```

(b) (2 pt) Write a predicate `quality-checker?` that takes in a product and returns true or false. It returns true if and only if

- its serial number contains at least one 0
- OR applying its instruction to its serial number produces its target number

Assume `instruction` always takes in a list and produces an integer result.

```
STk> (quality-checker? p1)
#t
STk> (define p2 (make-product 'cadr cadr 2 '(2 0 1 -1 8 -3)))
STk> (quality-checker? p2)
#t
```
3. (3 points) Data Directed Programming

Sam loves to drink water, so he has all sorts of water containers. Unfortunately, he’s having trouble keeping track of all the water he can drink since each container holds different amounts of water.

(a) (1 pt) There are three types of water containers: cup, bottle, and jug. A cup can hold 100mL of water, a bottle 500mL, and a jug 2000mL. Each type of container is a tagged list, tagged with its type and keeps track of a decimal of its percent full.

Here are the constructors for cups, bottles and jugs:

(define (make-cup percent)
  (attach-tag 'cup percent))
(define (make-bottle percent)
  (attach-tag 'bottle percent))
(define (make-jug percent)
  (attach-tag 'jug percent))

Write the calls to put to make selectors for the percent full for each container. Afterward, this should work:

(define (percent vessel)
  (operate 'percent vessel))
(define cup1 (make-cup 0.8))
(define bottle1 (make-bottle 0.4))
(define jug1 (make-jug 1))

STk> (percent cup1)
0.8
STk> (percent bottle1)
0.4

For reference, this is the operate procedure:

(define (operate op obj)
  (let ((proc (get (type-tag obj) op)))
    (if proc
      (proc (contents obj))
      (error "Unknown operator for type"))))

Now, write the calls to put to grab the amount of liquid in the container in mL. Afterward, these should work:

STk> (operate 'amount cup1)
80
STk> (operate 'amount bottle1)
200
STk> (operate 'amount jug1)
2000

Be careful not to violate the ADT!

(b) (2 pt) Write a procedure total-amount that takes in a list of containers and outputs the total amount of water the Sam has.

STk> (total-amount (list cup1 bottle1 jug1))
2280