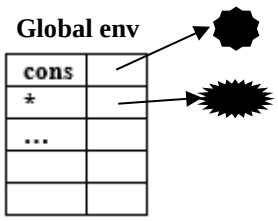
CSSE 304 Exam #2 E&C Oct 15, 2020 (day 23) Name_____

(25 points) Draw all of the environments and closures created during the execution of the code segment below. Include a sequence number for each environment or closure that you create and a number whenever something is added to the global environment. You do not need to do anything with variables that are already in the global environment. Comments are **not** required.

You can add to this document or do the problem on a blank page. Either way, submit your answer to the Exam 2 E&C drop box on Moodle.

Resources allowed: Only the things that are on the two pages of this document. You may not use a Scheme interpreter or look at any previous Environments and Closures examples.



A lot of the credit will be for correct environment pointers and correct sequence numbers.

```
(let* ([cons-c (lambda (x)
                 (lambda (y)
                   (cons x y)))]
       [uncurry2 (lambda (f)
                   (lambda (a b)
                     ((f a)b)))]
       [kons (uncurry2 cons-c)])
  (kons 3 4))
```

Environments and Closures summary

Environment:

var	val	refernce to a local environment containing variables from the enclosing let or lambda
var	val	
var	val	

Closure

list of formal parameter names	code (body of the procedure)	pointer to the local environment that was current when the closure was created

Create a procedure:
A user-defined procedure (a.k.a. **closure**) is created when a `lambda` expression is evaluated. The body of the procedure is not evaluated at this time.

Apply a closure (user-defined procedure):

1. The expressions for the procedure and its arguments are evaluated.
2. A new local environment is created.
   a. Each variable from the procedure's formal parameter list is bound to the corresponding value from the actual argument list.
   b. The new environment's "pointer to an enclosing environment" is set to be a copy of the local environment pointer **that is the third part of the closure**.
3. The body of the procedure is evaluated, using this new local environment. If a variable is not found in this local environment or something it points to, look in the global environment. If not in global environment either, it is an error.

Evaluate a `let` expression:

1. Evaluate (in the current environment) the expressions to get the values to be assigned to the `let` variables.
2. Create a new local environment that has bindings for the let variables. The "enclosing environment" pointer points to the current environment.
3. Evaluate the body of the `let` in this new environment, as in 3 above.

Evaluate a `letrec` expression:

1. Create a new local environment, similar to a `let` environment, except that:
   a. The "saved environment" pointers of any closures that are bound to the `letrec` variables point to the new letrec environment, not the enclosing environment.
2. Evaluate the body of the `letrec` in this new environment, as in 3 above.

Your diagram must follow the style used in the class examples:

1. A closure has **three parts** (argument list, code, environment pointer).
2. A local environment has **two parts** (a table of variables and their values, and a pointer to an environment produced by enclosing code (if any).
3. Environment pointers always point to environments, never to closures or code.
4. The value associated with a variable in an environment is never an environment, and it is never code.
5. You can't have two arrows coming from the same "pointer location".
6. Arrows never point to something "inside the box" of an environment or closure.
7. Place sequence numbers (start with 1) near each environment or closure that you draw and near each new entry in the global environment, to indicate the order in which these references are created during the execution of the code.
8. For simplicity in the case of `let`, you should pretend that `let` is executed directly (without translation into an application of `lambda`), so that all you need to show is the environment extension, rather than creating a closure followed by the environment extension that results from the application of that closure.
9. Show all changes this code makes to the global environment, and also include those in your sequence numbering.

Don't spend a lot of time trying to lay out the diagram in a neat way.
Do make sure that your scanned or photographed diagram is easy to read.