**CSSE 304 Assignment #15**

**Some past Piazza questions and answers**

# Is car not a primitive procedure?

1c has us create 1st-cps so that we can use it as an argument call. I thought we were allowed to use primitive procedures in non tail positions, and isn't 1st just a renaming of car?

**the students' answer** 1st-cps takes two arguments, a list and a continuation. Car just jakes a list. Once you start writing domain-cps you will see the distinction.

# HW15 set-of-cps member or member?

There is a "member" procedure in the given code of set-of. Do we treat it as a primitive procedure, or we need to substitute it with member?-cps?

(define set-of

    (lambda (s)

        (cond

            [(null? s) '()]

            [(***member*** (car s) (cdr s)) (set-of (cdr s))]

            [else (cons (car s) (set-of (cdr s)))])))

**the students' answer** You need to use member?-cps.

# tail position?

is the last argument to apply-continuation in tail position? for example in (apply-continuation k (some-cps 'x another-k)) is the call to some-cps in tail position?

**the students' answer,** I don't think it's technically in tail position since nothing in a function application is in tail position. **the instructors' answer** Not in tail position, so you are not allowed to write that code.

**followup discussions** Ok, so I see it's not in tail position then.  So you'd have to rearrange the code so that some-cps happens before that and apply the continuation to the result.

[**Claude Anderson**](https://piazza.com/class/is9cjqgxyh31b?cid=89) Yes.

# pair? in HW15

Could we just use pair? Directly? Or we have to write a pair?-cps instead?

**the students' answer** I believe pair? is a primitive procedure, so we can probably use it directly

# map-cps

Is the continuation taken by map-cps just passed to the cps-procedure? Wouldn't map-cps need a continuation for both the procedure and the list itself?

**the instructors' answer,** The continuation passed to map-cps is the continuation of the call to map-cps.  map-cps needs to create the continuations that it passes to the calls to the procedure being mapped.

map-cps is not a trivial procedure to write; my version is 10 lines long.

# Restriction on Identity procedure

The assignment says that if

"The continuation that you pass into one of your recursive calls is the identity procedure: (lambda (v) v)."

is a clue that your program may not be in proper CPS form.  Does this mean that within a function, we cannot call another cps function and give it the identity procedure?

For example, in set?-cps we can't do (member?-cps .... (lambda (v) v))?  
I'm just not sure how else we can get the value that we need since we'll be applying some continuation.

**the instructors' answer,** A call to a CPS procedure (like member?-cps) does not return a value. When it gets a value, it passes that value to its continuation.  That continuation must be able to do ALL of the rest of the computation.

If you think you need to pass the identity procedure as the continuation, I expect that your call to member?-cps is not in tail position, a violation of the "tail-recursive" rules.  You probably need to call member?-cps and include in the continuation of that call a call to whatever function you were going to pass the result of that identity procedure to.

# Applying a continutation

Is there a difference between doing

(apply-continuation k v)

and just doing

(k v)

?

**the instructors' answer,** The difference is representation-independence.  Don't forget that we are going to be changing our representation of continuations soon.  If you use apply-continuation in your code, you will not have to change it when we change the representation.  It will also make your code more readable.  And it will make the use of trace for debugging be more effective.

# CPS: Should make-k and apply-k be treated as primitives?

make-k: primitive

For now, it just wraps a lambda.  In a couple of weeks when we do data-structures representations of continuations,  make-k will just make a datatype object, no recursion involved.  Thus calls to make-k do not have to be (and almost never are) in tail position.

apply-k:  not primitive

When we apply a continuation, that may result in a call to a non-primitive procedure.  Thus all calls to apply-k must be in tail position.

# make-cps?

I tried testing map-cps but it said that make-cps is not bound. Was that a procedure that we had to write?

**the students' answer,**

*where students collectively construct a single answer*

It's part of 1d. It's one of those times where a test refers to something you haven't written yet.

**the instructors' answer,**

*where instructors collectively construct a single answer*

I could not find an easy way to avoid circularity in the dependence among the various procedures.

And why would we want make-cps except to produce procedures to feed to map?

# Is list a primitive

In terms of what needs to be in tail form is list, as in (list a 4 5) -> (a 4 5), a primitive procedure

**the instructors' answer,**

*where instructors collectively construct a single answer*

[Actions](https://piazza.com/class/jl863803n0a6tl?cid=98)

Yes

# "cps-snlist-recur is not itself a cps procedure...

, but it expects all of its arguments that are procedures to be cps procedures"

Does that mean when implementing cps-snlist-recur, all the rules of non-primitive, tail position calls go out the window?

**the students' answer,**

*where students collectively construct a single answer*

[Actions](https://piazza.com/class/jl863803n0a6tl?cid=99)

No.

The procedure returned by cps-snlist-recur should still be cps. You can always test your code by tracing all the non-primitive procedures.

I put the following lines at the end of my document. When testing on any of the test cases, it should print no indentation.

(trace apply-k)  
(trace andmap-cps)  
(trace make-cps)  
(trace domain-cps)  
(trace map-cps)  
(trace set-of-cps)  
(trace 1st-cps)  
(trace set?-cps)  
(trace member?-cps)  
(trace +-cps)  
(trace cps-snlist-recur)  
(trace sn-list-sum-cps)  
(trace sn-list-depth-cps)  
(trace sn-list-occur-cps)  
(trace sn-list-reverse-cps)

**the instructors' answer,**

*where instructors collectively construct a single answer*

[Actions](https://piazza.com/class/jl863803n0a6tl?cid=99)

The procedures that **cps-snlist-recur**takes as arguments, must be cps.  And also the procedure that it returns is cps.

# A15 Problem 3

Just in case other people have this problem. with-values is not included in scheme library. There is define-syntax code for with-values under call-with-values. Hope this helps.

PS: call-with-values takes procedures, not valu