This brief set of notes answers the question:
What does the "explicit control" evaluator have to do with the one that we discussed at the beginning of our consideration of interpreters?
("Interpreter" and "Evaluator" mean pretty much the same thing.)

Imagine two procedures:

Ceval (eval with continuation)
eval (the regular, initial one)

Here is the correspondence between them: for "any" 1-argument function K (our "continuation"), and any Scheme expression E, and any environment env,

(K (eval E env)) = (Ceval E env K)

[I won't put quotes around the E, etc. — but they are there!!]

The proof of this equality is by **induction** on the expression E. ("Induction" is just the mathematician's way of talking about what we call "recursion".)

I won't do all the cases: I'll just do some of the base cases and some important and general expressions, where you can see the use of the inductive hypothesis.

EXAMPLES:

(start with the BASIS of this INDUCTIVE ASSERTION!)

Constants

(Ceval C env K) = (K C) = (K (eval C env))

Variables

(Ceval x env K) = (K (lookup x env))

(on to the INDUCTIVE STEP…)

Conditionals

(Ceval (if P C A) env K) = ?? (K (Eval (if P C A) env)) ?? [I write ?? to mean "what we're trying to prove"]

(Ceval P env (lambda (p) (Ceval (if p C A) env K)))

by inductive hypothesis (because P is a subexpression of (if P C A) — read "smaller" — so we can apply induction to this smaller case…)

= ((lambda (p) (Ceval (if p C A) env K)) (eval P env))
= (Ceval (if p C A) env K)

When p=#t we have

(Ceval C env K) = (K (eval C env))

and similarly for p=#f…

Application

```
(Ceval ((lambda (v) B) E) env K) = ?? (K (eval ((lambda (v) B) E) env)) ??

= (Ceval (lambda (v) B) env
           (lambda (f)
             (Ceval E env
               (lambda (a) (f a K))))))   [by definition of application]

= (Ceval (lambda (v) B) env
           (lambda (f)
             ((lambda (a) (f a K)) (eval E env))))   [by inductive hypothesis
on E]

= (Ceval (lambda (v) B) env
           (lambda (f) (f (eval E env) K))  [substitution model]

= ((lambda (f) (f (eval B env) K))   [by definition of lambda]
     (lambda (x k)
       (Ceval E (extend v x) k)))

= ((lambda (x k) (Ceval B (extend v x) k)) (eval E env) K)

= (Ceval B (extend v (eval E env)) K)   [substitution model]

= (K (eval B (extend v (eval E env))))      [by inductive hypothesis on E]

= (K ((lambda (v) B) E) env)   [have a good look at the metacircular
evaluator in the book!]
```

The proof breaks down when we look at enter and exit, etc. … but for the "purely functional" stuff, it works fine.

IF YOU READ AND UNDERSTAND THE CASE FOR APPLICATION, then you'll see why the definitions of lambda and application are as in the handout on explicit continuations. You need to understand this to do the last problem set!

This induction proof may look complicated—it has a bit of detail—but it really isn't any more complicated than the similar proof for factorial in the notes on explicit continuations. Let Cfactorial be the version of factorial with an explicit continuation: induction on integers lets us prove for any integer m and continuation (one-argument function) K that

```
(Cfactorial m K) = (K (factorial m))
```

We're doing the same thing here—just for eval, instead of factorial.