**CSSE 304 Assignment #19 (individual assignment)**

**This is an individual assignment, not a partner assignment. You may discuss ideas with others and get help with debugging, but the code you write should be your own.**

**Programming part (80 points)**

Convert the CPS code that I provide to imperative form. In imperative form,

* All calls to non-primitive procedures must be in tail position.
* All non-primitive procedures must be thunks (procedures that are called with no arguments).

You will need to convert the lambda CPS to data structure CPS — that is, you should use something like (define-datatype continuation continuation? … in your final code. In your continuation datatype, you must provide the following continuations that contain no fields: id-k, list-k, length-k. I also recommend another one, r-f-p-k, which can be used by read-flatten-print. read-flatten-print will not be called by my test code, but it may be the most convenient approach for debugging your code. I used it to debug my code.

All of the continuation constructors may be considered primitive, but apply-k is not primitive, so it must be a thunk that is only called in tail position.

Other non-primitives include flatten-cps, list-sum-cps, the helper procedure in cps-snlist-recur, +cps, append-cps, cons-cps.

You must use cps-snlist-recur to produce the flatten-cps and sum-cps procedures, but cps-snlist-recur itself is not a CPS procedure; it takes CPS arguments and returns a CPS procedure

Code that gets all of the points from the server but does not follow the above rules will receive reduced credit, possibly zero credit.

This starting code contains many comments and test code that I hope you will find useful:

<http://www.rose-hulman.edu/class/csse/csse304/201920/Homework/Assignment_19/A19-starting-code.ss>

**Interface for grading purposes:**

Our tests on the server and offline will not use read-flatten-print. Instead they will assume that you have defined global variables slist and k, and they will call flatten-cps as in the following example:

(begin (set! slist '(((a d () (e) c ) g b) t))

(set! k (list-k))

(flatten-cps))

The other tests will have the same format, but different initial values for slist and k. When k is init-k, apply-k should simply return the value of v.

In addition, some of the test-cases will call sum-cps, which you will also create using cps-snlist-recur:

(begin

(set! slist '((1) () 2 (3)))

(set! k (id-k))

(sum-cps))