1 2017 Paper 23 Question 4

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2017p23q4.pdf

Consider the following simple evaluator for a language of expressions written in OCaml.

```
type expr =
        | Integer of int
        | Pair of expr * expr
        | Apply of string * expr

type value =
        | INT of int
        | PAIR of value * value

let rec eval = function
        | Integer n     -> INT n
        | Pair (e1, e2) -> PAIR (eval e1, eval e2)
        | Apply (f, e)  -> eval_function(f, eval e)
```

In this code the function `eval_function` has type `string * value -> value` and is used to evaluate some "built in" functions. For example,

```
eval_function("add", PAIR(INT 10, INT 7))
```

could return the value `INT 17`.

(a) Rewrite the `eval` function in continuation apssing style (CPS) to produce a function `eval_cps` so that the function

```
let eval_2 e = eval_cps (fun x -> x) e
```

will produce the same result as the function `eval`.

```
let rec eval_cps k = function
        | Integer n     -> k (INT n)
        | Pair (e1, e2) -> eval_cps (fun x -> eval_cps (fun y -> k (PAIR (x, y)) e2))
        | Apply (f, e)  -> eval_cps (fun x -> k (eval_function (f, x))) e
```

(b) Eliminate higher-order continuations from your `eval_cps` function. That is, introduce a data type `cnt` to represent continuations and write functions of type

```
eval_cps_dfn    : cnt -> expr -> value
apply_cnt       : cnt * value -> value
eval_3          : expr -> value
```

using the technique of defunctionalisation. Note that `eval_cps_dfn` and `apply_cnt` will be mutually recursive.

```
type cnt = Nil
        | Eval_pair_cnt of expr * cnt
        | Make_pair_cnt of value * cnt
        | Apply_cnt of string * cnt

let rec eval_cps_dfn k = function
        | Integer n -> apply_cnt (k, INT n)
        | Pair (e1, e2) -> eval_cps_dfn (Eval_pair_cnt(e2, k)) e1
        | Apply (f, e) -> eval_cps_dfn (Apply_cnt(f, k)) e

and apply_cnt = function
        | Nil, v -> v
```

```
        | Eval_pair_cnt(e, k), v -> eval_cps_dfn (Make_pair_cnt(v, k)) e
        | Make_pair_cnt(v1, k), v2 -> apply_cnt (k, (PAIR(v1, v2)))
        | Apply_cnt(f, k), v -> apply_cnt (k, eval_function(f, v))

let eval_3 = eval_cps_dfn Nil
```