Y2017P3Q4. CPS & Defunctionalisation

```
                                                      (* apply a named function: ADD *)
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)

let v = eval e in
  c(eval_function(f, v))
```

1. Add a continuation parameter c to each function, return value.

(a)

```
let rec eval_cps c = function
|Integer n ->  c ( INT n )
|Pair (e1,e2) -> eval_cps ( fun v1 -> (* PAIR1 *) λv2,e2,c      let v1 = eval e1 in
                   eval_cps ( fun v2 -> (* PAIR2 *) λv1,v2,c      v2 = eval e2 in
                      c (PAIR(v1,v2))) e2)                        c (PAIR(v1, v2))
                 e1)
|Apply (f, e) -> eval_cps (fun v -> c(eval_function(f, v)) e
                           (* FUNC *) - λf.c.
```

Thus

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

(b) Eliminate high-order continuations.

1. Add a constructor to ~~f~~ cnt for each fun (* Cnti *) λ... (free variables)

```
  type cnt =
|ID
|PAIR1 of expr * cnt
|PAIR2 of value * cnt
|FUNC of string * cnt
```

Call apply_cnt at every application of continuation.

```
let rec eval_cps_dfn c = function
|Interger n -> apply_cnt c (INT n)
|Pair (e1, e2) -> eval_cps_dfn ($PAIR1 (e2,C)) e1
|Apply (f, e) -> eval_cps_dfn (FUNC (f, c)) e

and apply_cnt = function
|( ID , V ) -> V
|(PAIR1 (e1, C), V1) -> eval_cps_dfn (PAIR2(V1,C)) e1
|(PAIR2 (V1, C), V2) -> apply_cnt ( C, PAIR (V1, V2 ))
|(Func(f;c), V) -> apply_cnt ( c, eval_function (f, V))

let eval_3 e = eval_cps_dfn ID e .
```

Mutually recursive.