y2017p3q4. CPS & Defunctionalisation

ADD_PAIR : PAIR → value.
ADD_INT : INT → value

(* apply a named function *)

```
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)
```

INT

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 *) λV1,e2,C
                    let V1 = eval e1 in
                    V2 = eval e2 in
                    C (PAIR (V1, V2))
                  eval_cps (fun V2 → (* PAIR2 *) λV1,V2,C
                    C (PAIR (V1,V2)) e2)
                  e1)
| Apply (f, e) → eval_cps (fun V → C (eval_function (f, V))) e ..... apply_cnt
                    (* FUNC *) λf,C.
```

Thus

```
let eval_2 e = eval_cps (fun x → x) e .
```

(* ID *) ... part b.

eval : expr → value.

eval_cps : fun expr → value

(IH) C (eval e) = eval_cps C e

(b) Eliminate high-order continuations.

1. Add a constructor to cnt for each fun (* CAStic *) 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
| Integer n → apply_cnt C (INT n)                          ×0
| Pair (e1, e2) → eval_cps_dfn (PAIR1 (e2, C)) e1          ..
| Apply (f, e) → eval_cps_dfn (FUNC (f, C)) e              ×1

and apply_cnt = function
| (ID, V) → V
| (PAIR1 (e2, C), V1) → eval_cps_dfn (PAIR2 (V1, C)) e2     ×2
| (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.

fun V1 ↦ eval_cps ... e2
fun V2 ↦ C (PAIR (V1, V2))
fun V ↦ C (eval_function (f, V))

eval_cps_dfn : cnt → expr → value   (for all)
apply_cnt : cnt * value → value
eval_3 : expr → value