4 Compiler Construction (TGG)

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

```
type expr =
  | Integer of int (* integer *)
  | Pair of expr * expr (* pair *)
  | Apply of string * expr (* apply a named function *)

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

(* eval : expr -> 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 passing 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 results as the function `eval`. [10 marks]

(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 functions `eval_cps_dfn` and `apply_cnt` will be mutually recursive. [10 marks]