PROGRAMMAZIONE II - a.a. 2016-17
Sesta esercitazione — 1 dicembre 2016

Esercizio 1 [luglio 2014]. Si estenda il linguaggio didattico funzionale in modo da includere espressioni e valori di tipo **record**. Un *valore (di tipo) record* è un dato strutturato composto da un numero finito di coppie *nome-valore*, detti **campi**. Analogamente, una *espressione (di tipo) record* è composta da un numero finito di coppie *nome-valore*. La valutazione di una espressione record produce un valore record.

Un identificatore può esser legato a un valore record tramite il costrutto `let`: nel seguente esempio (in sintassi OCaml-like) si evidenzia che l'espressione record che compare nel `let` è valutata in un valore record

```
# let rect = record{base = 5 * 5, altezza = 10 - 6}
val rect = record{base = 25, altezza = 4}
```

Sui record è definita l'operazione di selezione di una componente. Continuando l'esempio precedente

```
# let b = rect.base
val b = 25
```

1. Si estenda la sintassi astratta del linguaggio didattico funzionale in modo da includere valori ed espressioni record e l'operatore di selezione.
2. Si estenda il codice OCaml dell'interprete per trattare la valutazione di espressioni record e dell'operatore di selezione.

Si veda il file `main.ml` *per una implementazione minimale.*

Esercizio 2 [gennaio 2015]. Si consideri il seguente programma OCaml

```
let n = 5;;
let h = fun x -> n + x;;
let rec f p n =
   let g = fun y -> n * y in
      if n = 0 then p 1
      else if n > 1 then f g (n-1)
           else f p (n-1);;
f h 2;;
```

1. Si indichi il tipo inferito dall'interprete OCaml per la funzione ricorsiva `f`.

```
val n : int = 5
val h: int -> int = <fun>
val f: (int -> int) * int -> int = <fun>
```

2. Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione.

   Si segua lo svolgimento alla lavagna. :-)

3. Si indichi il valore restituito dal programma.

```
- : int = 2
```

Soluzione esercizio 1

```
(* Espressioni *)

type ide = string;;

type exp = Int of int
         | Den of ide
         | Let of ide * exp * exp
         | Record of (ide * exp) list
         | Select of exp * ide;;

type dexp = DInt of int
         | DRec of (ide * dexp) list;;

(* run time *)

exception EmptyEnv;;

(* Ambiente locale default *)
let env0 = fun x -> raise EmptyEnv;;

(* Estensione di ambiente *)
let ext env (x: ide) v = fun y -> if x=y then v else env y

(*
env : string -> 'a
ext : (string -> 'a) * string * 'a -> string -> 'a
*)

(* valutazione di espressioni
   NB:  booleani sono rappresentati da 0 (false) e 1 (true) *)

let rec eval e env = match e with
     Int i -> DInt i
   | Den x -> env x
   | Let (x, e1, e2) -> let v = (eval e1 env) in
                            let env1 = (ext env x v) in
                                (eval e2 env1)

   | Record e1 -> let v = (evalList e1 env) in DRec v
   | Select (e1, i) -> match e1 with
                            Den x -> (match env x with
                                         DRec c -> (lookup i c)
                                       | _ -> failwith "wrong ide in select")
                          | _ -> failwith "wrong exp in select"
and evalList el env = match el with
                             [] -> []
                           | (i,e)::el1 -> (i, (eval e env)):: (evalList el1
env)
and lookup x c = match c with
                     [] -> failwith "wrong field in select"
                   | (y, v)::c1 -> if x = y then v else lookup x c1;;

let dIntToInt de = match de with
                      DInt x -> x
                    | _ -> failwith "wrong int";;

print_int (dIntToInt (eval (Let("x",Record([("z", Int 7);("y",Int
5)]),Select(Den "x","y"))) env0));;
```