

Algorithmique-Programmation : IAP1 - contrôle continu CORRIGE -
Lundi 9 octobre 2006
Sans documents - durée : 30 minutes

Nom :

Prénom :

Compléter la session suivante. Il n'y pas d'erreur syntaxique mais des erreurs de typage peuvent exister. Dans ce cas, indiquez la réponse "erreur de typage" et expliquez en quelques mots pourquoi..

Répondez directement sur la feuille.

```
1 #let f x = x +. x;;
```

```
val f : float -> float = <fun>
```

```
2 #let g (x, y) = y && (x>0);;
```

```
val g : int * bool -> bool = <fun>
```

```
3 #g (2, true);;
```

```
- : bool = true
```

```
4 #g (true, 2);;
```

```
This expression has type bool * int but is here used with type int * bool
```

```
5 # a - a;;
```

```
Unbound value a
```

```
6 #let m (c,e) = match c with (0,_) -> e
                           |(x, true) -> x
                           | _ -> failwith "m : Pb";;
```

```
val m : (int * bool) * int -> int = <fun>
```

```
7 #m (1, false, 10);;
```

```
This expression has type int * bool * int but is here used with type
(int * bool) * int
```

```
8 #m ((0,true), 10);;
```

```
- : int = 10
```

```
9 #m ((1, false), 10);;
```

```
Exception: Failure "m : Pb".
```

```
10>(* Interface base_8
type : int -> int
arguments : n
precondition : n >=0
postcondition : base_8 n est le chiffre de poids le plus fort dans le
codage de n en base 8
*)
```

```
let rec base_8 n =
if n <8 then n
else base_8 (n /8);;
```

```

11 #(* Interface rr
type : int list -> bool
arguments : 1
precondition : vraie (ou aucune)
postcondition : teste si tous les éléments
                 de la liste sont pairs
*)
let rec rr l = match l with
  [] -> true
  | x::r -> (x mod 2 = 0) && (rr r);;

```

```

12 #rr [4;2;1;3];;
- : bool = false

```

13 Donner une autre version de la fonction rr: celle-ci devra utiliser une fonction récursive terminale. Vous écrirez avec soin l'interface de cette fonction récursive terminale.

```

(* interface rraux
type : int list * bool -> bool args : 1 acc
pre : aucune
post : retourne acc&&(rr l) *)
let rec rraux (l, acc) = match l with
  [] -> acc
  | x::r -> rraux (r, acc && x mod 2 = 0);;

let rr1 l = rraux (l, true);;

```

```

14 # type tt = A of string | B of int;;
# [ A "ab"; B 3];;

- : tt list = [A "ab"; B 3]

```

```

15 #(* Interface concat
type : tt list -> int*string
arguments : 1
précondition : vraie (ou aucune)
postcondition : concat l calcule simultanément la somme des entiers
contenus dans l et la chaîne résultant de la concaténation de toutes
les chaînes contenues dans l
tests concat [] (doit retourner (0,"")), concat [ A "ab"; B 3 ; B 6; A "toto"]
(doit retourner (9, "abtoto")), concat [B 4; B 10] (doit retourner (14, "")),
concat [A "aa"; A "10"] (doit retourner (0, "aa10")) ;;
*)
let rec concat l = match l with
  [] -> (0, "")
  | (A x)::r -> let (a,b) = concat r in (a, x^b)
  | (B x)::r -> let (a,b) = concat r in (a+x, b);;

```

```

16 # let toto c = (snd c, (fst c) + 1);;
val toto : int * 'a -> 'a * int = <fun>

```