Algorithmique-Programmation : AP1 - contrôle continu -
Lundi 11 octobre 2010 - Sans documents - durée : 35 minutes

Répondez directement sur la feuille.

CORRECTION

Nom : ..... **Prénom :** .....

Exercice 1

Compléter la session suivante. Il n'y a 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.
On rappelle que les réponses de Ocaml sont de la forme `-:type=valeur` ou `val id:type=valeur`. La valeur d'une fonction est notée `<fun>`. Vous respecterez cette syntaxe.

```
#let f b = if b > 1.4 then b +. 2. else b ;;
val f : float -> float = <fun>

#f 3;;
   ---
Error: This expression has type int but an expression
was expected of type float

#let g x = (x mod 2 = 0, x/2);;
val g : int -> bool * int = <fun>

#g 5;;
- : bool * int = (false, 2)

#let h (x,y) = (y, x+2);;
val h : int * 'a -> 'a * int = <fun>

#let b = h (0,5) in h b;;
- : int * int = (2, 7)

#b;;
Error: Unbound value b

#let k (a,e) = match e with
    [x] -> (x+a)/2
  | x::r -> a
  | [] ->  0;;
val k : int * int list -> int = <fun>

#[k (1, [7;3;2]) ; k (20 , [1]) ; k (20, [20]) ; k (30, [])];;
- : int list = [1; 10; 20; 0]

#let rec assoc1 (l,a)  = match l with
    [] -> failwith "Non trouvé"
  | (x,y)::r -> if x=a then y else assoc1 (r, a);;
```

```
val assoc1 : ('a * 'b) list * 'a -> 'b = <fun>
```

```
#assoc1 ([(1,"toto") ; (2, "titi") ; (3, "tata")], 4);;
```

```
Exception: Failure "Non trouvé".
```

```
#let rec assoc2 (l,a,e)  = match l with
    [] -> e
  | (x,y)::r -> if x=a then y else assoc2 (r,a,e);;
```

```
val assoc2 : ('a * 'b) list * 'a * 'b -> 'b = <fun>
```

```
#assoc2 ([(1,"toto") ; (2, "titi") ; (3, "tata")], 4, " ");;
```

```
- : string = " "
```

Exercice 2

Écrire l'interface des fonctions `qui_suis_je` et `qui_suis_je2` définies ci-dessous. Pour la post-condition, vous avez droit à une formulation informelle (en français ou en pseudo-mathématique). Vous ignorerez la clause `Tests` de l'interface.

```
(*interface qui_suis_je
type : int list -> int
args l
precondition : l contient au moins un entier positif
postcondition : retourne le premier entier positif de l
raises : lève l'exception Failure "échec" si la liste ne contient
aucun entier positif.
tests ne pas compléter
*)
let rec qui_suis_je l = match l with
    []-> failwith "échec"
  | x::r -> if x>0 then x else qui_suis_je r;;
```

```
(*interface qui_suis_je2
type : int list -> int list
args l
precondition : rien
postcondition : retourne la liste des entiers positifs contenus dans l
raises : aucune
tests ne pas compléter
*)
let rec qui_suis_je2 l = match l with
    []-> []
  | x::r -> if x>0 then x::(qui_suis_je2 r) else qui_suis_je2 r;;
```

Exercice 3

Ecrire une fonction qui retourne simultanément la somme et le produit d'une liste d'entiers. Lorsque la liste est vide, le résultat sera le couple (0,1).

```
let rec sp l = match l with
    []-> (0,1)
  | x::r -> let (s,p) = sp r in (x+s, x*p);;
```

Exercice 4 (**Commandes Unix**)

Dans cet exercice vous êtes l'utilisateur `toto`.

```
ls -l
total 0
-rw-r--r--  1 toto toto 3 Oct  6 17:41 f1
-r--r--r--  1 toto toto 5 Oct  6 17:41 f2
```

Expliquer pourquoi la commande `cp f1 f2` échoue.

On ne peut copier vers un fichier dont on n'a pas les droits en écriture. (cela peut dans certains cas s'outrepasser avec l'option -f).
Remarque : on ne peut copier depuis un fichier dont on n'a pas les droits en lecture.