Université Paris 6
Licence d'informatique

Module de Programmation
ANNÉE 2000-2001

Travaux Dirigés n° 12

Exercice 1 *(Récursion terminale et accumulateur)*
Rappel : on définit la fonction factorielle de la façon suivante

```
let rec fact n = if n <= 0 then 1 else n*(fact (n-1)) ;;
```

On peut donner une version *récursive terminale* de cette fonction en définissant une fonction récursive ayant un argument supplémentaire appelé *accumulateur*.

```
let rec fact_t a n =
  if n <= 0 then a else (fact_t (n*a) (n-1));;

let fact'n = fact_t 1 n;;
```

1 Montrer que pour tout `n` (positif ou nul) on a `(fact n)`=`(fact' n)` (par récurrence sur `n`).

2 La fonction `min` (de type `'a -> 'a -> 'a`) retourne le plus petit de ses arguments pour l'ordre polymorphe <=. On définit une fonction de recherche d'un élément minimal dans une liste (non vide) :

```
let rec lmin l = match l with
| [] -> failwith "lmin: []"
| [x] -> x
| x::l' -> (min x (lmin l')) ;;
```

Sur le principe utilisé ci-dessus, donnez une version récursive terminale de la fonction `lmin`.

3 Soit la fonction `reverse` de type `'a list -> 'a list`

```
let rec reverse l = match l with
| [] -> []
| x::l' -> (reverse l')@[x];;
```

Sur le principe utilisé ci-dessus, donnez une version récursive terminale `reverse2` de la fonction `reverse`.

4 Montrez que les fonctions `reverse` et `reverse2` sont équivalentes.

5 Comparez la complexité en temps et en espace de ces deux versions. Qu'en concluez-vous ?

Exercice 2 *(Accumulateurs et références)*
Voici une autre version de la factorielle.

```
let fact n =
  let a = ref 1 in
  let rec fact_r n =
    if n <= 0 then !a else begin a := n*(!a); fact_r (n-1) end
 in fact_r n;;
```

1 Expliquez l'évaluation de cette déclaration puis l'évaluation de l'expression `(fact 3)`.

2 Donnez une nouvelle version de `lmin` en utilisant ce dernier principe.

Exercice 3 *(Références et fermetures)*
Soit la suite de phrases

```
let f1 = function () ->
  let c = ref 0 in (c := !c+1; !c) ;;
f1 () ;;
f1 () ;;

let f2 =
  let c = ref 0 in function () -> (c := !c+1; !c);;
f2 () ;;
f2 () ;;
```

Détaillez l'évaluation de cette suite de phrases.

Exercice 4 *(Références et boucles)*

Voici encore une version de la factorielle !

```
let fact n =
  let a = ref 1 and i = ref 1 in
  let rec fact_b() =
    if n < (!i)
       then !a
       else begin a := (!i) * (!a); i := !i + 1; fact_b() end
 in fact_b() ;;
```

1 Détaillez l'évaluation de cette déclaration puis celle de l'expression `fact 3`.

2 Déduire de cette dernière déclaration de la factorielle une ultime implémentation utilisant une boucle `for`.