Université A.Mira Béjaia **Département Maths MI 1ière année**

Examen de Rattrapage - Calcul formel - 20 septembre 2008

Nom :	Groupe : .....
Prénom :	Emargement :

Exercice1 : (05 pts)

1. Définissez une variable a égale à (3x+1) où x vaut 3. Attention, x ne doit pas être définie globalement !

```
# let a = let x = 3 in 3*x+1;;
val a : int = 10
```

2. Définissez une fonction qui à l'entier x associe (3x+b) où b vaut localement 1.

```
# let f x = let b = 1 in 3*x+b;;
val f : int -> int = <fun>
```

3. Définissez une fonction g qui à l'entier x associe $(3x+1)^2$

```
# let g x = (3*x+1)*(3*x+1);;
val g : int -> int = <fun>
```

4. Même question qu'en 3. mais en ne calculant qu'une fois la valeur de (3x+1) et sans utiliser de fonction "carré".

```
# let g x = let a = 3*x+1 in a*a;;
val g : int -> int = <fun>
```

5. Même problème qu'à la question 4, mais en utilisant une fonction locale pour calculer le produit (3x+1).

```
# let g x = let f y = 3*y+1 in (f x)*(f x);;
val g : int -> int = <fun>
```

Exercice2 : (2,5 pts) Pour chacune des expressions suivantes, si elle est correcte dire son type et si elle est incorrecte, indiquer pourquoi ?

```
1. let p = fun a b p -> p a b ;;
2. let f x y=if x then x+y else x-y;;
3. let rec f = function
      [] -> []
      | t::r -> f r;;
```

```
val p : 'a -> 'b -> ('a -> 'b -> 'c) -> 'c = <fun>
```

This expression has type bool but is here used with type

```
val f : 'a list -> 'b list = <fun>
```

Exercice 3 : (2,5 pts)

1. Typer la fonction suivante

```
# let rec f = fun l -> match l with
   [] -> []
   |t::r -> l @ f r;;
```

```
val f : 'a list -> 'a list = <fun>
```

Que renvoie l'appel **f [1; 2; 3] ;;**

```
- : int list = [1; 2; 3; 2; 3; 3]
```

2. Typer l'expression suivante

```
#let rec f l =
  match l with
  [] -> []
  |t::r -> g t (f r)
  and g e l =
   match l with
   [] -> [e]
   |t :: r -> if e <= t then e :: l else t :: g e r;;
```

```
val f : 'a list -> 'a list = <fun>

val g : 'a -> 'a list -> 'a list = <fun>
```

Que renvoie l'appel **f["os";"as";"sos";"est";"etc."];;**

```
- : string list = ["as"; "est"; "etc."; "os"; "sos"]
```

Problème : (10 pts) **Attention : tous les filtrages doivent être exhaustifs !**

1. Nous voulons trois versions d'une fonction qui prend une paire d'entiers en argument et qui renvoie la somme des carrés des éléments de la paire.
   a. une version simple (sans filtrage)
   b. une version avec filtrage.

```
# let sum (a,b) = (a*a)+(b*b);;
val sum : int * int -> int = <fun>
```

```
# let sum = function
      (a*a)+(b*b);;
val sum : int * int -> int = <fun>
```

2. Ecrivez une fonction qui prend en entrée une liste et qui renvoie la somme des deux premiers entiers la constituant.

```
# let somme = function
   [] -> failwith "pas assez d'arguments"
   |[a] -> failwith "pas assez d'arguments"
   |a::b::r -> a+b;;
val somme : int list -> int = <fun>
```

```
# let somme l = match l with
a::b::r -> a+b
|_-> failwith "pas assez d'arguments";;
val somme : int list -> int = <fun>
```

3. Ecrivez une fonction qui prend en entrée les coordonnées de deux points (sous forme de paires de réels) et qui renvoie la longueur du segment correspondant.

```
# let longueur (a,b) (c,d) = sqrt((a-.c)*.(a-.c) +. (b-.d)*.(b-.d));;
val longueur : float * float -> float * float -> float = <fun>
Où :
# let longueur a b = match (a,b) with
  ((x,y),(z,t)) -> sqrt((x-.z)*.(x-.z) +. (y-.t)*.(y-.t));;
val longueur : float * float -> float * float -> float = <fun>
```

4. Ecrivez une fonction qui prend en argument une paire de listes et qui renvoie la somme du premier élément de la première liste et du premier élément de la deuxième.

```
# let paire_de_listes = function
      (a::_,b::_) -> a+b
      | _ -> failwith "une des deux listes est vide";;
val paire_de_listes : int list * int list -> int = <fun>
```

5. Ecrire une fonction qui retourne la valeur du maximum d'une liste.

```
# let rec maximum = function
      [] -> failwith "Liste vide"
      |[a] -> a
      | t::r -> max t (maximum r);;
val maximum : 'a list -> 'a = <fun>
```

6. Ecrire une fonction qui prend une liste en entrée et retourne la liste privée de son élément maximal.

Avec la fonction maximum :

```
# let sansmax l =
      let rec ote x = function
      t::r -> if (t=x) then r
            else t::(ote x r)
      |[] -> []
      in ote (maximum l) l;;
val sansmax :'a list->'a list=<fun>
```

Sans la fonction maximum :

```
# let sansmax =
  let rec aux x = function
  [] -> []
  |t::r ->if (t>x) then (x::(aux t r))
      else t::(aux x r)
  in function [] -> []
              |t::r -> aux t r;;
val sansmax :'a list ->'a list=<fun>
```