TD de programmation fonctionnelle et logique
Corrigé du TD 4 : typage et polymorphisme

Expressions à typer

```
# let f x y = x;;
```

```
# let f x y = y;;
```

```
# let f x y = if true then x else y;;
```

```
# let f x y = if false then x else y;;
```

```
# let f x y = if x=y then 0 else 1;;
```

```
# let f x y = if x=y then x else 1;;
```

```
# let f x y = if x=y then x+y else x-y;;
```

```
# let f x y = if x then x+y else x-y;;
```

```
# let f g x = g(g x);;
```

```
# let rec f a b c =
   match a with
       []   -> [b c]
     | t::r  -> (b t)::(f r b c);;
```

```
# let f x y = x;;
 val f : 'a -> 'b -> 'a = <fun>
```

```
# let f x y = y;;
 val f : 'a -> 'b -> 'b = <fun>
```

```
# let f x y = if true then x else y;;
 val f : 'a -> 'a -> 'a = <fun>
```

```
# let f x y = if false then x else y;;
 val f : 'a -> 'a -> 'a = <fun>
```

```
# let f x y = if x=y then 0 else 1;;
 val f : 'a -> 'a -> int = <fun>
```

```
# let f x y = if x=y then x else 1;;
 val f : int -> int -> int = <fun>
```

```
# let f x y = if x=y then x+y else x-y;;
 val f : int -> int -> int = <fun>
```

```
# let f x y = if x then x+y else x-y;;
 Characters 22-23:
This expression has type bool but is here used with type int
```

```
# let f g x = g(g x);;
```

```
val f : ('a -> 'a) -> 'a -> 'a = <fun>
```

```
#let rec f a b c =
   match a with
       []   -> [b c]
     | t::r  -> (b t)::(f r b c);;
       val f : 'a list -> ('a -> 'b) -> 'a -> 'b list = <fun>
```

Addition polymorphe des éléments d'une liste

1. Écrivez une fonction récursive qui calcule la somme des éléments d'une liste d'entiers de taille supérieure ou égale à un.

```
#let rec somme = function
     [a] -> a
   | t::r -> t + (somme r)
   |  _    -> failwith "Liste vide interdite";;
       val somme : int list -> int = <fun>
```

2. Écrivez une fonction récursive qui calcule la somme des éléments d'une liste de taille supérieure ou égale à un, quel que soit le type des éléments de cette liste.

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

3. Typez la fonction précédente.
4. Écrivez une fonction récursive qui applique récursivement une opération binaire (+, `*.`, `^`, etc.) aux éléments d'une liste de taille supérieure ou égale à un, quel que soit le type des éléments de cette liste.
   La solution de la question 2 répond à la présente question.
5. Même question que précédemment mais en ne supposant plus que la liste soit forcément non vide.

```
#let rec somme add e = function
     [] -> e
   | t::r -> add t (somme add e r);;
     val somme : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b = <fun>
```

6. Typez la fonction précédente.