Programmation fonctionnelle et logique
TD2 : typage et récursivité

Expressions à typer

1. ```
   let f x y = x;;
   val f : 'a -> 'b -> 'a = <fun>
   ```
2. ```
   let f x y = y;;
   val f : 'a -> 'b -> 'b = <fun>
   ```
3. ```
   let f x y = if true then x else y;;
   val f : 'a -> 'a -> 'a = <fun>
   ```
4. ```
   let f x y = if false then x else y;;
   val f : 'a -> 'a -> 'a = <fun>
   ```
5. ```
   let f x y = if x=y then 0 else 1;;
   val f : 'a -> 'a -> int = <fun>
   ```
6. ```
   let f x y = if x=y then x else 1;;
   val f : int -> int -> int = <fun>
   ```
7. ```
   let f x y = if x=y then x+y else x-y;;
   val f : int -> int -> int = <fun>
   ```
8. ```
   let f x y = if x then x+y else x-y;;
   This expression has type bool but is here used with type int
   ```

Petits programmes récursifs

1. Écrire une fonction récursive qui prend en entrée deux tableaux (supposés être de même taille) et qui vérifie si un tableau est le miroir de l'autre (ex. : `[|5;3;4;6|]` est le miroir de `[|6;4;3;5|]`).

   ```
   let miroir a b  =
     let rec miroir_aux rang =
         if rang = (Array.length a)
            then true
            else a.(rang)=b.((Array.length a)-1-rang) & miroir_aux (rang+1)
     in
      miroir_aux 0;;
   val miroir : 'a array -> 'a array -> bool = <fun>
   ```

2. Écrire une fonction récursive qui prend en entrée deux tableaux (supposés être de même taille) et qui recopie le premier tableau dans le deuxième.

   ```
   let recopie origine destination =
     let rec recopie_aux rang =
         if rang = (Array.length destination)
            then destination
            else begin
                   destination.(rang) <- origine.(rang);
                   recopie_aux (rang+1)
   ```

```
              end
  in
   recopie_aux 0;;
val recopie : 'a array -> 'a array -> 'a array = <fun>
```