UE - Programmation Fonctionnelle

TD 4. Les modules

Exercice 1 : Interface :

```
module type collection_interface =
sig
  type 'a t
  exception collectionException;
  val estVide : 'a t -> boolean
  val vide : 'a t
  val ajouter : 'a -> 'a t -> 'a t
  val enlever : 'a t -> ('a * 'a t)
end.
```

```
module type collection =
struct
  type 'a t;
  val ajoute : 'a t;
  val enveler : 'a t;
  val estVide : 'a t -> boolean
  val vide : 'a t
```

```
module pile : collection_interface =
struct
  type 'a t = 'a ~~tist~~ list
  exception collectionException
  let estVide e l = (l = [])
  let vide e = []
  let ajouter e l = e :: l
  let enlever l = match l with
    | [] -> raise collectionException
    | hd :: tl -> (hd :: tl)
end
```

```
module File : collection_interface
struct
  let ajouter e col =
    col @ [e]
end.
```

测试 :

```
let % test_ =
```

Exercice 2 : 例 : fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b

Interface :

```
module type fold =
sig
  type a
  type b
  val cas_terminal : b
  val traite_et_combine :
      a -> b -> b
end
```

```
module type fold =
sig
  type a
  type b
  val cas_terminal : b
  val traite_et_combine :
      a -> b -> b
end
```

```
module creat_list : fold =
struct                    <with type a = int and type b = int list>
  type a = int
  type b = int list
  let cas_terminal = []
  let traite_et_combine e l =
      e :: l
end
```

```
type 'a option =
| None
| Some of 'a
```

```
module find_pair : fold =
struct                    <with ...>
  type a = int
  type b = int option
  :
  let cas_terminal = None
  let traite_et_combine e b
    if e mod 2 = 0 then Some e
      else b
end.
```

```
module FoldList (F: fold) =
struct
  let rec foldright l = match l with
                        | [] -> F.cas_terminal
                        | hd::tl -> F.traite_et_combine t
                                    (foldright tl)
end
module FoldCreateList = FoldList (creat_list)
module FoldCollection (C: collection_interface) (F: fold)
struct
  let rec fold col = if C.est_vide col then F.cas_terminal
                     else let (e, col2) = C.enlever col in
                          F.traite_et_combine e (fold col2)
module FoldPile = FoldCollection (Pile)
module CreatListPile = FoldPile (CreatList)
let % test_ = CreatListePile.fold (Pile.(ajouter 1 (ajouter 2 vide)))
            = [1;2]
```