DS n°3

TD et TP

T1.
```
let points main = Array.fold_left
  (fun acc carte -> acc + (points_carte carte))
  0
  main
;;
```

T2.
```
let creer_fileb capacite init = {
  taille = 0;
  tete = 0;
  queue = 0;
  donnees = Array.make capacite init
};;

let enfiler_fileb f x =
  let capacite = Array.length f.donnees in
  if f.taille = capacite then
    failwith "File pleine"
  else
    f.donnees.(f.queue) <- x;
    f.queue <- (f.queue + 1) mod capacite;
    f.taille <- f.taille + 1
;;

let defiler_fileb f =
  let capacite = Array.length f.donnees in
  if f.taille = 0 then
    failwith "File vide"
  else
    let x = f.donnees.(f.tete) in
    f.tete <- (f.tete + 1) mod capacite;
    f.taille <- f.taille - 1;
    x
;;

let est_vide_fileb f = f.taille = 0;;
```

T3.
```
let rec map func l = match l with
  | [] -> []
  | x::q -> (func x)::map func q
;;
```

T4.
```c
void tri_denombrement(int t[], int taille, int k) {
        int *counts = malloc((k+1)*sizeof(int));
        for (int i=0; i<k+1; i+=1) {
            counts[i] = 0;
        }
        for (int i=0; i<taille; i+=1) {
            counts[t[i]] += 1;
        }
        int i = 0;
        for (int j=0; j<k+1 && i < taille; j+=1) {
            while (counts[j] > 0) {
                t[i] = j;
                i += 1;
                counts[j] -= 1;
            }
        }
    }
```

Exercices

Exercice 1

Première fonction :

```
let rec somme l = match l with
  | [] | _::[] -> l
  | (x,a)::(y,b)::q ->
    if x = y then
      (x,a+.b)::somme q
    else
      (x,a)::somme ((y,b)::q)
;;
let func1 l = somme (List.fast_sort compare l);;
```

Deuxième fonction :

```
(* On suppose que l est une liste de Hashtbl *)
let rec func2 l = match l with
  | [] -> Hashtbl.create 0
  | ht::q ->
    let ht2 = func2 q in
    Hashtbl.iter
      (fun cle x ->
        Hashtbl.replace ht2 cle (
          match Hashtbl.find_opt ht2 cle with
            | None -> x
            | Some x2 -> x+.x2
        )
      )
      ht;
    ht2
;;
```

Troisième fonction :

```
struct couple {
    int cle;
    double val;
};
typedef cpl;
void sort(cpl t[], int taille) {
    if (taille <= 1) {
        return;
    }
    int pivot = t[taille-1].cle;
    int i = 0;
    int j = taille-1;
    while (i<j) {
        while (t[i].cle<=p && i<j) i+=1;
        while (t[j].cle>=p && i<j) j-=1;
        cpl temp = t[i];
        t[i] = t[j];
        t[j] = temp;
    }
    sort(t, i+1);
    sort(t+i+1, taille-i-1);
}
// Renvoie la nouvelle taille du tableau
int func3(cpl t[], int taille) {
    sort(t, taille);
    int i=0;
    for (int j=0; j<taille-1; j+=1) {
        t[i] = t[j];
        while (j<taille-1 && t[j].cle == t[j+1].cle) {
            t[i] += t[j+1].val;
            j+=1;
        }
        i+=1;
    }
    if (taille > 1 && t[taille-2].cle != t[taille-1].cle) {
        t[i] = t[taille-1];
        i+=1;
    }
    return i;
}
```