Feuille de révisions n°2 - Programmation en OCaml : manipulation de tableaux

1 Pied à l'étrier

R. 2-1 Écrire une fonction calculant l'indice du premier `0` dans un tableau d'entiers parcouru par boucle `while`. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
let find_indice_first_0 (t: int array): int =
  let n = Array.length t in
  let i = ref 0 in
  while (!i < n && t.(!i) <> 0) do
    incr i;
  done;
  if !i = n then -1 else !i
```

R. 2-2 Écrire une fonction calculant l'indice du premier `0` dans un tableau d'entiers parcouru par boucle `for`, arrêtée au moyen d'une levée d'exception. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
exception Found of int
let find_indice_first_0_bis (t: int array): int =
  let n = Array.length t in
  try
    for i = 0 to (n-1) do
      if t.(i) = 0 then raise (Found(i))
    done; -1
  with
  | Found(i) -> i
```

R. 2-3 Écrire une fonction calculant l'indice du **dernier** `0` dans un tableau d'entiers parcouru par boucle `while`. La fonction devra retourner `-1` si le tableau ne contient aucun `0`.

Solution

```
let find_indice_last_0 (t: int array): int =
  let n = Array.length t in
  let i = ref (n-1) in
  while (!i >=0 && t.(!i) <> 0) do
    i := !i-1;
  done;
  !i (* if !i = -1 then -1 else !i *)
```

R. 2-4 Écrire une fonction testant l'existence d'un `0` dans un tableau d'entiers avec une boucle `while`, arrêtée dès que possible sans utiliser d'exception.

Solution

```
let existe_0 (t: int array): bool =
  let res = ref false in
  let i = ref 0 in
  while (!i < Array.length t && not !res) do
    if t.(!i) = 0 then res := true else ();
    i := !i+1;
  done;
  !res
```

R. 2-5 Écrire une fonction testant l'existence d'un `0` dans un tableau d'entiers avec une fonctionnelle `Array.iter`, arrêtée par une levée d'exception.

Solution

```
exception Found
let existe_0_bis (t: int array): bool =
  try  Array.iter (fun x -> if x = 0 then raise Found) t; false
  with | Found -> true
```

R. 2-6 Écrire une fonction prenant en argument un tableau d'entiers et calculant la somme des éléments du tableau. On fournira une implémentation au moyen d'une boucle `for`.

Solution

```
let somme_tab (t: int array) : int =
  let res = ref 0 in
  let n = Array.length t in
  for i = 0 to (n-1) do
    res := !res + t.(i)
  done;
  !res
```

R. 2-7 Écrire une fonction prenant en argument un tableau d'entiers et calculant la somme des éléments du tableau. On fournira une implémentation au moyen d'un `Array.fold_left`.

Solution

```
let somme_tab2 (t: int array) : int =
  Array.fold_left (fun acc x -> acc + x) 0 t
```

R. 2-8 Écrire une fonction calculant l'indice du minimum dans un tableau d'entiers parcouru par une boucle `for`. Cette fonction déclenchera l'exception `Invalid_argument` dans le cas où le tableau est de taille `0`.

Solution

```
let indice_minimum (t: int array): int =
  let n = Array.length t in
  if n = 0 then raise (Invalid_argument "tableau de taille 0")
  else
    let i_min = ref 0 in
```

```
    let v_min = ref t.(0) in
    for i = 1 to (n-1) do
      if t.(i) < !v_min then
        begin
          i_min := i;
          v_min := t.(i)
        end
    done;
    !i_min
```

R. 2-9 Écrire une fonction calculant l'indice du maximum dans un tableau d'entiers positifs parcouru par une fonctionnelle `Array.fold_left`. Cette fonction déclenchera l'exception `Invalid_argument` dans le cas où le tableau est de taille `0`.

Solution

```
let indice_maximum (t: int array): int =
  let aux = (fun (i_max, v_max, i) _ ->
    if t.(i) > v_max then (i, t.(i), i+1) else (i_max, v_max, i+1))
  in let i_max, _, _ = Array.fold_left aux (-1, -1, 0)  t
  in i_max

```

R. 2-10 Écrire une fonction prenant en argument un tableau d'entiers `t` et calculant le tableau des sommes cumulées du tableau `t`. Ainsi la case d'indice `i` du tableau résultat doit contenir `t.(0) + t.(1) + ... + t.(i)`. On fournira une implémentation en $\mathcal{O}(n)$.

Solution

```
let tab_some_cumulee (t: int array): int array =
  let n = Array.length t in
  if n = 0 then [||]
  else
    begin
      let rep = Array.make n t.(0) in
      for i = 1 to (n-1) do
        rep.(i) <- t.(i) + rep.(i-1)
      done;
      rep
    end
```

R. 2-11 Définir une fonction prenant en argument un tableau de listes d'entiers `t` et le modifiant en ajoutant à chaque liste l'entier indice de la case du tableau dans laquelle se trouve la liste. On itérera sur le tableau au moyen d'une boucle `for`

Solution

```
let ajoute_indice_listes_bis (t: int list array): unit =
  let n = Array.length t in
  for i = 0 to (n-1) do
    t.(i) <- i :: t.(i)
```

```
  done
```

R. 2-12 Définir une fonction prenant en argument un tableau de listes d'entiers `t` et un entier `x` et retournant un tableau de listes d'entiers obtenu par ajout de l'entier en tête de chacune des listes se trouvant dans le tableau. On itérera sur le tableau au moyen d'un `Array.map`.

Solution

```
let ajoute_indice_listes(t: int list array) (x: int): int list array =
  Array.map (fun l -> x :: l) t
```

R. 2-13 Écrire une fonction prenant en argument un tableau d'entiers `t`, contenant des valeurs entières dans un intervalle $[\![0, m-1]\!]$ où m vous est passé en argument, et calculant l'entier de $[\![0, m-1]\!]$ ayant le plus d'occurrences dans le tableau. On s'efforcera d'itérer sur les tableaux aux moyens de fonctionnelles et non de boucle `for`/`while`. On fournira une implémentation en $\mathcal{O}(n+m)$.

Solution

```
let element_majoritaire (t: int array) (m: int): int =
  let tab_occurs = Array.make m 0 in
  Array.iter (fun x -> tab_occurs.(x) <- tab_occurs.(x) + 1) t;
  let aux = (fun (i_max, v_max, i) x -> if x > v_max  then (i, x, i+1)
                                        else (i_max, v_max, i+1))
  in let i_max, _, _  =  Array.fold_left aux (-1, -1, 0) tab_occurs
  in i_max
```

2 Tris

R. 2-14 Écrire une fonction permettant de trier un tableau au moyen de l'algorithme du tri à bulles♣.

R. 2-15 Écrire une fonction permettant de trier un tableau au moyen de l'algorithme du tri par insertion.

R. 2-16 Écrire une fonction permettant de trier un tableau au moyen de l'algorithme du tri par sélection.

R. 2-17 Écrire une fonction partition prenant en argument un tableau `t` et `ip` un indice de `t` et qui modifie `t` en place de manière à placer d'abord les éléments inférieurs à la valeur pivot `t[ip]`, puis cette valeur pivot, à l'indice `q` qu'elle retournera, et enfin ceux strictement supérieurs à cette valeur.

R. 2-18 Écrire une fonction permettant de trier un tableau au moyen de l'algorithme du tri rapide.

♣. https://fr.wikipedia.org/wiki/Tri_à_bulles