

Devoir surveillé d'informatique

⚠ Consignes

- Les programmes demandés doivent être écrits en C ou en OCaml. Dans le cas du C, on suppose que les bibliothèques standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`, `<stdassert.h>`, ...) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veuillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

□ Exercice 1 : Tris

On rappelle ci-dessous en pseudo-langage, l'algorithme du tri par sélection (aussi appelé tri par recherche itérée du minimum)

Algorithme : Tri par sélection

Entrées : tableau $tab = \{t_0, \dots, t_{n-1}\}$ de n éléments

Sorties : aucune, le tableau tab est modifié et trié en place

```

1  pour  $i \leftarrow 0$  à  $n - 1$  faire
2       $i_m \leftarrow$  indice du minimum du sous tableau  $\{t_i, \dots, t_{n-1}\}$ 
3      échanger dans  $t$  les éléments d'indice  $i$  et  $i_m$ 
4  fin

```

Q1– En recopiant et complétant le tableau suivant. Donner l'état des variables de l'algorithme lorsqu'on utilise cet algorithme pour trier le tableau $\{2, -5, -4, 4, 1\}$

	i_m	tab
après un tour de boucle	1	$\{-5, 2, -4, 4, 1\}$
après deux tours de boucle	2	$\{-5, -4, 2, 4, 1\}$
après trois tours de boucle	4	$\{-5, -4, 1, 4, 2\}$
après quatre tours de boucle	4	$\{-5, -4, 1, 2, 4\}$

Q2– Ecrire en langage C une fonction de signature `indice_min_depuis(int tab[], int n, int ind)` qui renvoie l'indice d'une occurrence du minimum du tableau tab de taille n à partir de l'indice ind (inclus).

```

1  int indice_min_depuis(int tab[], int taille, int ind)
2  {
3      assert(ind < taille && ind >= 0);
4      int vmin = tab[ind];
5      int imin = ind;
6      for (int i = ind; i < taille; i++)
7      {
8          if (tab[i] < vmin)
9          {
10             imin = i;
11             vmin = tab[i];
12         }
13     }
14     return imin;
15 }

```

Q3– En langage C, on suppose déjà écrite une fonction de signature `void echange(int tab[], int i, int j)` qui échange les éléments situés aux indices i et j dans le tableau tab . Ecrire une fonction de signature

`tri_selection(int tab[], int n)` qui tri le tableau `tab` de taille `n` en utilisant l'algorithme du tri par sélection.

```

1 void tri_selection(int tab[], int size)
2 {
3     int im;
4     for (int i = 0; i < size; i++)
5     {
6         im = indice_min_depuis(tab, size, i);
7         echange(tab, i, im, size);
8     }
9 }

```

Q4– Montrer que l'algorithme du tri par sélection a une complexité quadratique en fonction de la taille n du tableau.

La fonction de recherche du minimum a une complexité en $\mathcal{O}(n)$ car la boucle de recherche s'exécute au plus n fois et elle ne contient que des opérations élémentaires. Le tri appelle n fois la fonction de recherche du minimum et donc a une complexité en $n\mathcal{O}(n) = \mathcal{O}(n^2)$.

Q5– On suppose qu'on a trié un tableau de taille 500 000 en 1.2 secondes avec l'algorithme du tri par sélection, donner une estimation du temps nécessaire pour trier un tableau contenant un million d'éléments.

La taille du tableau est doublé et la complexité est quadratique donc le temps de calcul sera approximativement multiplié par 4, on doit donc s'attendre à un temps de calcul d'environ 4.8 secondes.

Q6– Ecrire en OCaml une fonction `separe : int list -> int list* int list` qui prend en argument une liste d'entiers `lst` et renvoie un couple de deux listes contenant chacune une moitié des éléments de `lst` (à une unité près). Par exemple `separe [2; 8; 1; 4; 5]` peut renvoyer le couple `[2; 1; 5]`, `[8; 4]`.

```

1 let rec separe l =
2     match l with
3     | [] -> [], []
4     | h::[] -> [h], []
5     | h1::h2::t -> let l1,l2 = separe t in h1::l1, h2::l2;;

```

Q7– Ecrire en OCaml une fonction `fusion : int list -> int list -> int list` qui prend en argument deux listes d'entiers triées et renvoie la liste triée de leur fusion. Par exemple `fusion [1; 2; 5] [4; 8]` renvoie la liste `[1; 2; 4; 5; 8]`.

```

1 let rec fusion l1 l2 =
2     match l1, l2 with
3     | [], _ -> l2
4     | _, [] -> l1
5     | h1::t1, h2::t2 -> if h1<h2 then h1::(fusion t1 l2) else h2::(fusion l1 t2);;

```

Q8– On rappelle que l'algorithme du tri fusion est un algorithme de tri récursif qui consiste en séparer en deux la liste des éléments à trier, à trier récursivement chacune de ces listes puis à les fusionner. Dédurre des questions précédentes une fonction `tri_fusion : int list -> int list` qui prend en argument une liste d'entiers et renvoie cette liste triée.

```

1  let rec tri_fusion l =
2    match l with
3    | [] -> []
4    | h::[] -> [h]
5    | _ -> let l1,l2 = separe l in
6            let tl1 = tri_fusion l1 in
7            let tl2 = tri_fusion l2 in
8            fusion tl1 tl2;;

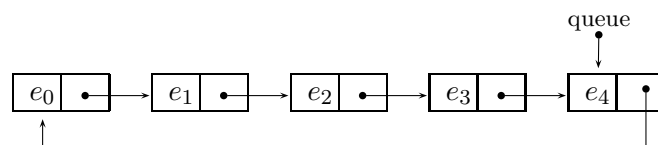
```

- Q9**– On admet qu'il faut n comparaisons pour fusionner deux listes dont la somme des longueurs est n . On cherche à déterminer la complexité du tri fusion en *nombre de comparaison effectuées* et on note $C(n)$ le nombre de comparaisons afin de trier une liste de longueur n . Déterminer $C(0)$, $C(1)$ et justifier rapidement que $C(n) = 2C(\frac{n}{2}) + n$.
- Q10**– Dédurre de la relation précédente que pour tout $i \in \mathbb{N}^*$, $C(n) = 2^i C(\frac{n}{2^i}) + in$ et en déduire la complexité du tri fusion en nombre de comparaisons.

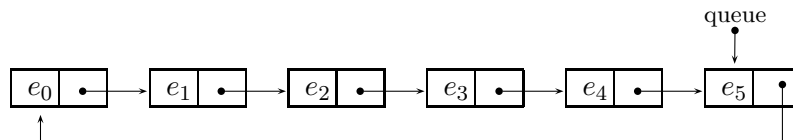
□ **Exercice 2 : Liste chaînée circulaire**

Le langage utilisé dans cet exercice est le langage C.

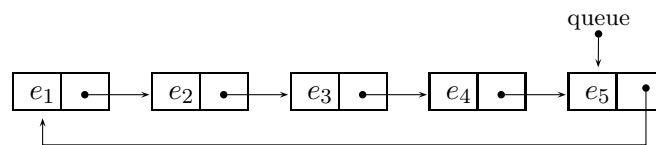
On veut implémenter une structure de données de liste chaînée circulaire avec un pointeur sur la queue telle que représentée ci-dessous :



La queue pointe toujours vers *le dernier élément inséré* ainsi, après l'ajout d'un nouvel élément e_5 , la structure de données ci-dessus devient :



Lorsqu'on retire un élément de cette structure de données, on retire le maillon qui *suit le pointeur de queue*. Par conséquent, si on retire un élément de la structure de donnée ci-dessus, c'est le maillon contenant e_0 qui est retiré et on obtient :



Afin d'implémenter cette structure de données, on propose d'utiliser les types suivants

```

1  struct maillon_s
2  {
3      int valeur;
4      struct maillon_s *suivant;
5  };
6  typedef struct maillon_s maillon;
7  typedef maillon *liste_circulaire;

```

La liste chaînée circulaire vide est alors représentée par le pointeur NULL.

- Q11**– En partant d'une liste chaînée circulaire initialement vide, donner les étapes de son évolution (en schématisant l'état de la liste), après les opérations suivantes :
1. ajouter 12
 2. ajouter 6

3. ajouter 7
4. retirer
5. ajouter 42
6. retirer

On précisera la valeur des entiers renvoyés par la fonction `retirer`.

```

1. ajouter 12 : 12
2. ajouter 6 : 12 → 6
3. ajouter 7 : 12 → 6 → 7
4. retirer : 6 → 7, l'opération renvoie 12
5. ajouter 42 : 6 → 7 → 42
6. retirer : 7 → 42, l'opération renvoie 6

```

Q12– Ecrire la fonction `ajouter` de signature `void ajouter(liste_circulaire *lc, int v)` qui modifie la liste circulaire donnée en argument en lui ajoutant un nouveau maillon contenant la valeur `v`.

🔗 Indication : on fera attention à traiter le cas particulier d'une liste circulaire initialement vide.

```

1 void ajouter(liste_circulaire *lc, int v)
2 {
3     maillon *nm = malloc(sizeof(maillon));
4     nm->valeur = v;
5     if (est_vide(*lc))
6     {
7         nm->suivant = nm;
8     }
9     else
10    {
11        nm->suivant = (*lc)->suivant;
12        (*lc)->suivant = nm;
13    }
14    *lc = nm;
15 }

```

Q13– Ecrire la fonction `retirer` de signature `int retirer(liste_circulaire *lc)` qui prend en argument une liste circulaire *supposée non vide* et renvoie la valeur du maillon situé après le pointeur de queue en le retirant de la liste circulaire.

```

1 int retirer(liste_circulaire *f)
2 {
3     int res = ((*f)->suivant)->valeur;
4     maillon *old = (*f)->suivant;
5     if ((*f)->suivant == *f)
6     {
7         *f = NULL;
8     }
9     else
10    {
11        (*f)->suivant = ((*f)->suivant)->suivant;
12    }
13    free(old);
14    return res;
15 }

```

- Q14–** Quelle structure de données connue a-t-on implémenté ici ? Justifier et proposer des noms plus appropriés pour les fonctions **ajouter** et **retirer**.

Les premiers éléments ajoutés à la structure sont aussi les premiers à être retirés, il s'agit donc d'une structure de données de type FIFO, c'est à dire une file. L'opération ajouter correspond à enfiler et l'opération retirer à défiler.

- Q15–** Ecrire une fonction **longueur** de signature `int longueur(liste_circulaire lc)` qui renvoie le nombre d'éléments d'une liste chaînée circulaire.

```

1  int longueur(liste_circulaire lc)
2  {
3      if (est_vide(lc))
4      {
5          return 0;
6      }
7      else
8      {
9          liste_circulaire start = lc;
10         int nb = 1;
11         while (lc->suivant != start)
12         {
13             nb += 1;
14             lc = lc->suivant;
15         }
16         return nb;
17     }
18 }
```

- Q16–** Donner, en les justifiant, les complexités des opérations **retirer**, **ajouter** et **longueur**.

Retirer et ajouter sont en $O(1)$ car on ne fait que des opérations élémentaires et longueur est en $O(n)$ où n est la longueur de la liste car celle ci doit être parcourue en entier (on détecte un retour vers le pointeur initial).

□ Exercice 3 : dépouillement d'un vote

Le langage utilisé dans cet exercice est OCaml. On s'intéresse à la comptabilisation des résultats d'une élection. Chaque candidat est désigné par un caractère ('A', 'B', ...) et on suppose qu'on dispose des votes sous la forme d'une liste de caractère, par exemple la liste ['A'; 'C'; 'B'; 'B'; 'A'; 'A'] signifie qu'il y a eu 6 votants, le candidat 'A' a obtenu 3 voix, le 'B' deux voix et le 'C' une voix. On souhaite dans un premier temps à partir de la liste des votes construire la liste des couples (candidat, nombre de votes), c'est à dire que si la liste des votes est ['A'; 'C'; 'B'; 'B'; 'A'; 'A'] alors on veut construire la liste [('A',3); ('C',1); ('B',2)].

- Q17–** Ecrire une fonction **appartient** : `char -> char list -> bool` qui prend en argument un caractère `c` et une liste de caractères `lst` et qui renvoie `true` si et seulement si `c` est dans `lst`.

```

1  let rec appartient c votes =
2      match votes with
3      | [] -> false
4      | h::t -> h=c || appartient c t;;
```

- Q18–** En déduire une fonction **candidats** : `char list -> char list` qui prend en argument une liste `votes` et renvoie cette liste sans aucun doublon, par exemple `candidats ['A'; 'C'; 'B'; 'B'; 'A'; 'A']` renvoie la liste ['A', 'C', 'B'].

```

1 let rec candidats votes =
2   match votes with
3   | [] -> []
4   | h::t -> let cr = candidats t in if appartient h cr then cr else h::cr;;

```

Q19– Ecrire une fonction `nb_votes : char -> char list -> int` qui prend en argument un caractère `c` et une liste de caractères `votes` et renvoie le nombre d'apparition de `c` dans `votes`. Par exemple `nb_votes 'A' ['A'; 'C'; 'B'; 'B'; 'A'; 'A']` renvoie 3.

```

1 let rec nb_votes c votes =
2   match votes with
3   | [] -> 0
4   | h::t -> if h=c then 1 + nb_votes c t else nb_votes c t;;

```

Q20– En déduire une fonction `comptabilise : char list -> (char*int) list` qui répond au problème posé, c'est à dire qu'elle prend en argument une liste de caractères `votes` et renvoie une liste de couples `(char*int)` de chacun des caractères apparaissant dans `votes` et de son nombre d'occurrences dans `votes`.

```

1 let comptabilise votes =
2   let cd = candidats votes in
3   List.map (fun c -> (c,nb_votes c votes)) cd;;

```

Q21– On veut maintenant écrire une version de la fonction `comptabilise` qui parcourt une seule fois la liste des votes. Pour cela on veut écrire une fonction `ajoute : char -> char*int list -> char*int list` qui prend en argument un caractère `un_vote`, une liste de couples `(char*int) list` qui représente un résultat partiel du dépouillement des votes et renvoie le nouveau décompte des voix après prise en compte de `un_vote`. Par exemple :

- `ajoute 'B' [('A',2); ('C',1)]` renvoie `[('A',2); ('C',1); ('B', 1)]` (premier vote obtenu par le candidat 'B')
- `ajoute 'C' [('A',2); ('C',1)]` renvoie `[('A',2); ('C',2)]` (incrémente le nombre de vote du candidat 'C')

. Ecrire la fonction `ajoute`.

```

1 let rec ajoute un_vote resultat =
2   match resultat with
3   | [] -> [(un_vote, 1)]
4   | (v,r)::t -> if v=un_vote then (v, r+1)::t else (v,r)::(ajoute un_vote t)

```

Q22– En déduire une nouvelle version de la fonction `comptabilise` qui parcourt une seule fois la liste des votes.

```

1 let rec comptabilise2 votes acc =
2   match votes with
3   | [] -> acc
4   | h::t -> comptabilise2 t (ajoute h acc);;

```

□ Exercice 4 : Représentation d'ensemble d'entiers

En OCaml, on propose de représenter un ensemble d'entiers, par la liste *triée* (dans l'ordre croissant) de ses éléments. Par exemple l'ensemble `{7; 2; 5; 3}` sera représenté par la liste `[2; 3; 5; 7]`. Par contre, les listes `[2; 3; 3; 5; 7]` (élément en double) ou `[2; 5; 3; 7]` (non triée) ne représentent pas correctement un ensemble.

Q23– Ecrire une fonction `est_ensemble: int list -> bool` qui renvoie `true` lorsque la liste d'entier fournie en argument représente correctement un ensemble.

```

1  let rec est_ensemble l =
2      match l with
3      | [] -> true
4      | h::[] -> true
5      | h1::h2::t -> h1<h2 && est_ensemble (h2::t);;

```

- Q24–** Ecrire une fonction `intersection : int list -> int list -> int list` qui prend en argument deux listes d'entiers (en supposant que ces deux listes représentent correctement des ensembles) et renvoie la liste d'entiers représentant l'union de ces deux listes. Par exemple `intersection [2; 5; 7] [5; 6; 7; 10];;` renvoie `[5; 7]`.

```

1  let rec intersection ens1 ens2 =
2      match ens1, ens2 with
3      | [], ens2 -> []
4      | ens1, [] -> []
5      | e1::r1, e2::r2 -> if e1=e2 then e1::(intersection r1 r2) else
6                          if e1<e2 then intersection r1 ens2 else
7                          intersection r2 ens1;;

```

- Q25–** Quelle serait la complexité d'une fonction `appartient : int -> int list -> bool` qui teste si un entier appartient ou non à une liste d'entiers représentant correctement un ensemble? (on ne demande pas d'écrire la fonction `appartient` mais simplement d'en justifier la complexité)

La complexité serait linéaire car dans le pire des cas, on doit parcourir toute la liste afin de savoir si l'élément cherché s'y trouve ou non.

On souhaite maintenant de façon similaire représenter un ensemble d'entier en langage C par le tableau trié de ses éléments, cette structure de donnée est *non mutable* c'est à dire que si on souhaite par exemple ajouter un élément à un ensemble alors on crée un nouvel ensemble.

- Q26–** Proposer un type structuré `ensemble` en C qui réalise cette représentation.
- Q27–** Ecrire la fonction de signature `bool appartient(ensemble ens, int elt)` qui renvoie `true` si et seulement si `elt` appartient à `ens`. Cette fonction *doit procéder* en parcourant un à un les éléments du tableau représentant l'ensemble. On donnera la complexité de cette fonction (aucune justification n'est demandée).
- Q28–** Donner en pseudo-langage un algorithme vu en cours et qui permettrait d'écrire une version de complexité linéarithmique de la fonction `appartient` en s'appuyant sur le fait que les éléments du tableau sont triés.