

## 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	...	...
après deux tours de boucle	...	...
après trois tours de boucle	...	...
après quatre tours de boucle	...	...

**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). On vérifiera les conditions sur  $ind$  à l'aide d'instructions `assert`.

**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.

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

**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.

**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]`.

**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]`.

**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

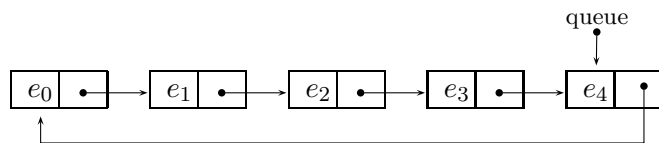
des questions précédentes une fonction `tri_fusion : int list -> int list` qui prend en argument une liste d'entiers et renvoie cette liste triée.

- Q9**– On admet qu'il faut au plus  $n$  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  $C(n) = 2C\left(\frac{n}{2}\right) + n$ .
- Q10**– Dédurre de la relation précédente que pour tout  $i \in \mathbb{N}^*$ ,  $C(n) = 2^i C\left(\frac{n}{2^i}\right) + in$  et en déduire la complexité du tri fusion en nombre de comparaisons.

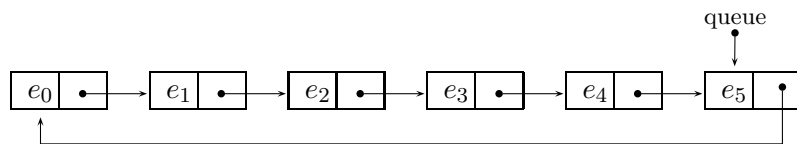
□ **Exercice 2 : Liste chaînée circulaire**

Le langage utilisé dans cet exercice est le langage C.

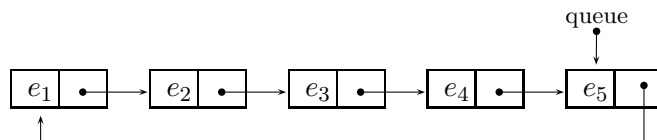
On veut implémenter une structure de données de liste chaînée circulaire avec un pointeur sur la queue telle que représentée ci-dessous :



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`.

- 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.
- 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. Une solution entièrement correcte devra libérer l'espace mémoire alloué au maillon que l'on supprime.
- Q14**– Quelle structure de données connue a-t-on implémenté ici ? Justifier et proposer des noms plus appropriés pour les fonctions `ajouter` et `retirer`.
- Q15**– Ecrire une fonction `longueur` de signature `int longueur(liste_circulaire lc)` qui renvoie le nombre d'éléments d'une liste chaînée circulaire.
- Q16**– Donner, en les justifiant, les complexités des opérations `retirer`, `ajouter` et `longueur`.

### □ 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`.
- 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'].
- 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.
- 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`.
- 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`.
- Q22**– En déduire une nouvelle version de la fonction `comptabilise` qui parcourt une seule fois la liste des votes.

### □ 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. La liste vide [] représente l'ensemble vide.

- Q23**– Ecrire une fonction `est_ensemble` : `int list` -> `bool` qui renvoie `true` lorsque la liste d'entier fournie en argument représente correctement un ensemble. Par exemple `est_ensemble [2; 8; 1]` doit renvoyer `false` car la liste n'est pas triée dans l'ordre croissant et `est_ensemble [1; 2; 7; 7; 10]` doit aussi renvoyer `false` car 7 est présent deux fois.
- 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'intersection des deux ensembles représentés par ces listes. Par exemple `intersection [2; 5; 7] [5; 6; 7; 10];;` renvoie `[5; 7]`.
- Q25**– *Sans les écrire*, donner les complexités des fonctions suivantes en les justifiant brièvement :
- `minimum` : `int list` -> `int` qui renvoie le plus petit élément d'un ensemble représenté par une liste d'entiers (on suppose l'ensemble non vide)
  - `maximum` : `int list` -> `int` qui renvoie le plus grand élément d'un ensemble représenté par une liste d'entiers (on suppose l'ensemble non vide)

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**– Donner la complexité des fonctions de signature `int minimum(ensemble ens)` et `int maximum(ensemble ens)` qui renvoie respectivement le minimum et le maximum de l'ensemble `ens` donné en argument. On ne *demande pas* d'écrire ces fonctions.
- Q28**– 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).
- Q29**– Donner en pseudo-langage un algorithme vu en cours et qui permettrait d'écrire une version de complexité logarithmique de la fonction `appartient`. Cette version doit s'appuyer sur le fait que les éléments du tableau sont triés.
- Q30**– Proposer une implémentation de cet algorithme en langage C.

```
1  bool dichotomie(ensemble ens, int elt)
2  {
3      int a = 0;
4      int b = ens.cardinal - 1;
5      int m;
6      while (b - a >= 0)
7      {
8          m = (a + b) / 2;
9          if (ens.elements[m] == elt)
10         {
11             return true;
12         }
13         else
14         {
15             if (ens.elements[m] > elt)
16             {
17                 b = m - 1;
18             }
19             else
20             {
21                 a = m + 1;
22             }
23         }
24     }
25     return false;
26 }
```