Devoir maison n°2

Ce devoir maison est à faire sur feuille et à rendre pour le 2 mars (au début de votre TP).

Exercice 1 *Tri par sélection*

Cet exercice est à faire en C.

Lorsqu'on effectue le tri croissant d'un tableau t, on peut se rendre compte que la plus petite valeur du tableau doit aller dans la case 0, la deuxième plus petite valeur dans la case 1, etc... Le principe du tri par sélection est donc de créer un nouveau tableau et de le remplir en trouvant le 1er minimum de t, le deuxième minimum de t, etc...

Pour un tableau d'entiers t de taille n, le pseudo-code est le suivant :

- Créer un tableau t' de taille n (de contenu quelconque)
- Créer une copie t_2 de t.
- Calculer le maximum $maxi$ du tableau t.
- Pour i allant de 0 à $n-1$:
  - Trouver le minimum $mini$ de t_2 et son indice $imin$.
  - Mettre $mini$ dans la case i de t'.
  - Mettre $maxi$ dans la case $imin$ de t_2.
- Renvoyer t'.

1. Écrire une fonction `int* copie(int* t, int n)` qui renvoie une copie du tableau t. **Attention : les deux tableaux doivent être indépendants**
2. Écrire une fonction `int indice_min(int* t, int n)` qui renvoie **l'indice du minimum** (et pas sa valeur). Par exemple pour $t = [2, 4, 1, 3]$, le minimum est dans la case 2, donc la fonction renverra 2.
3. En suivant le principe ci-dessus, écrire une fonctions `int* tri_selection(int* t, int n)` qui renvoie un tableau trié contenant les mêmes éléments que t.
4. Quelle est la complexité de cette méthode en temps ? En mémoire ?

On va maintenant effectuer le tri par sélection **en place**, c'est à dire en modifiant le tableau initial et sans créer de tableau auxiliaire.

Le principe est le suivant :

- Pour i allant de 0 à $n-1$:
  - Trouver le plus petit élément de t entre les indices i et $n-1$. On note $mini$ sa valeur et $imin$ son indice.
  - Échanger le contenu des cases i et $imin$: $mini$ va dans la case i et la valeur qui était dans la case i va dans la case $imin$.

5. Écrire une fonction `int indice_min_partiel(int* t, int n, int i)` qui renvoie **l'indice du minimum** (et pas sa valeur) en ne considèrant que les éléments après l'indice i (inclus).
   Par exemple pour $t = [5, 0, 4, 1, 10]$ et $i = 2$, la fonction renvoie 3 car le plus petit élément en ignorant les indices 0 et 1 est le 1, en position 3.
6. En suivant le principe ci-dessus, écrire une fonctions `void tri_selection_en_place(int* t, int n)` qui trie t par effet de bord (en retournant void).

Exercice 2 *Reconstruction d'arbres*

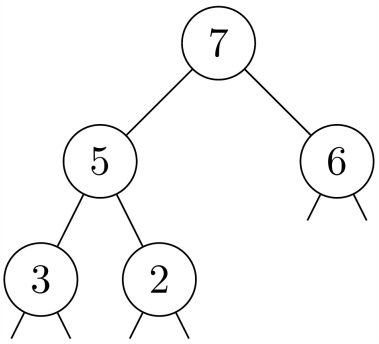
Cet exercice est à faire en Ocaml.

Un arbre binaire strict est un arbre binaire dans lequel les noeuds ont soit 0, soit 2 fils (et jamais un seul). L'arbre vide n'est pas considéré comme étant un arbre binaire strict.

Lors du parcours d'un arbre binaire strict, on construira une liste contenant le type suivant (F pour une feuille, NI pour un noeud interne)

```
type 'a noeud = F of 'a | NI of 'a;;
```

Par exemple le parcours préfixe de l'arbre suivant donne la liste [NI 7 ; NI 5 ; F 3 ; F 2 ; F 6] :



Le but de cet exercice est de reconstruire l'arbre à partir de la liste de ses noeuds, trouvée par un des parcours d'arbres.

1. Donner deux arbres distincts dont le parcours infixe produit la liste [F 0 ; NI 1 ; F 2 ; NI 3 ; F 4]. Que peut-on en conclure ?
2. Retrouver un arbre dont le parcours préfixe donne [NI 1 ; NI 2 ; F 4 ; F 5 ; F 4]. En existe-t-il un autre (aucune justification attendue) ?
3. Expliquer sur des exemples comment reconstruire un arbre à partir de sa liste de noeuds/feuilles pour un parcours préfixe. On pourra utiliser une pile.
4. Programmer l'idée précédente en Ocaml. La signature sera : `reconstruit : 'a noeud list -> 'a arbrebin`. Si vous avez besoin d'une pile, vous pouvez utiliser le module `Stack`.
5. Qu'est-ce qui changerait pour notre algorithme si on considère plutôt un parcours suffixe ?
6. Programmer en Ocaml un algorithme qui fait la même chose mais pour les listes données par le parcours en largeur. Si vous avez besoin d'une file, vous pouvez utiliser le module `Queue`.

Exercice 3 *Entiers de Péano*

Cet exercice est à faire en OCaml

On considère le type `entier` suivant qui représente les éléments de $\mathbb{N}$.

```
type entier =
  | Zero
  | Successeur of entier;;
```

Par exemple, l'entier 2 sera représenté par `let deux = Successeur ( Successeur Zero)`.

1. Écrire une fonction `int_vers_entier : int -> entier` qui transforme un entier donné par sa représentation OCaml en un entier avec notre représentation comme successeur de successeurs de 0. Par exemple, `deux` doit être égal à `int_vers_entier 2`
2. Écrire une fonction `entier_vers_int : entier -> int` qui réalise la transformation inverse. Par exemple, on pourra vérifier que `entier_vers_int deux = 2`
3. Écrire une fonction `add : entier -> entier -> entier` réalisant l'addition de deux entiers. On pourra remarquer que si $n \geqslant 1$ et $m \geqslant 0$, $n + m = 1 + ((n-1) + m)$. Il s'agit bien d'implémenter directement l'addition sur ce type et pas d'utiliser la fonction précédent pour repasser par les entiers de OCaml.
4. Écrire de même une fonction `mult` réalisant la mulitiplication de deux entiers, de type `entier -> entier -> entier`. Comme pour la question précédente, on n'utilisera pas la fonction `entier_vers_int`.