Dans tout ce TP le type `'a btree` fait référence au type OCaml ci-dessous défini.

```ocaml
type 'a btree =
| Empty
| Node of 'a * 'a btree * 'a btree
```

Exercice 1 : Tas binaire en OCaml

On définit le type `type path = bool list` permettant la représentation d'un chemin dans un arbre binaire.

Q. 1 Définir une fonction `next : path -> path` telle que `(next p)` donne la position suivant `p` pour l'ordre du parcours par niveau de gauche à droite dans un arbre binaire.

Exemples :

```
next [] = [false]
next [false] = [true]
next [true] = [false; false]
next [false; false] = [false; true]
next [true; true] = [false; false; false]
```

Q. 2 Définir une fonction `prev : path -> path` telle que, pour tout chemin `p` de type `path` on a : `prev (next p) = p`.

Dans le reste de cet exercice on implémente le type de donnée file de priorité au moyen des tas binaires présentés en classe. On définit pour cela le type `type 'a tas_bin = {tree : 'a btree; next_free : path}`.

Q. 3 Définir la fonction `mininim : 'a tas_bin -> 'a`, permettant la lecture du minimum dans une file de priorité.

Q. 4 Définir la fonction `insertion : 'a tas_bin -> 'a -> 'a tas_bin`, permettant l'insertion d'un élément dans une file de priorité.

Q. 5 Définir la fonction `suppression : 'a tas_bin -> 'a tas_bin`, permettant la suppression du minimum dans une file de priorité.

Exercice 2 : Tas binaire en C

On définit en C, le type ci-dessous permettant la représentation d'arbre binaire.

```c
typedef struct node_s {
  int valeur;
  struct node_s *pere;
  struct node_s *gauche;
  struct node_s *droit;
} * node;
```

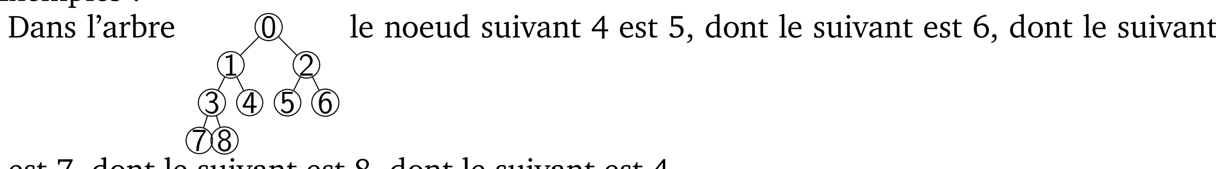
Ce type est le même que celui des arbres binaires du TP11, vous êtes invités à récupérer les différentes fonctions déjà implantées sur ce type qui peuvent vous être utile dans cet exercice.

Comme dans l'exercice précédent[♣], il est important de garder en mémoire la position d'insertion du prochain élément dans l'arbre. Pour se faire, on tire profit des pointeurs C et on garde en mémoire un pointeur vers le noeud en dessous duquel le prochain élément doit être inséré. On sauvegarde aussi un booléen, indiquant si la prochaine insertion doit se faire en fils gauche (`false`) ou en fils droit (`true`) du noeud pointé.
La structure complète a donc le type :

```c
typedef struct {
  node root;
  node father_next;
  bool lr;
} bheap;
```

Q. 1 Définir la fonction `node next(node cur)`, prenant en argument un noeud `cur` que l'on supposera être une feuille et retournant la prochaine feuille dans l'ordre de parcours, par niveau, de gauche à droite des feuilles d'un arbre complet.

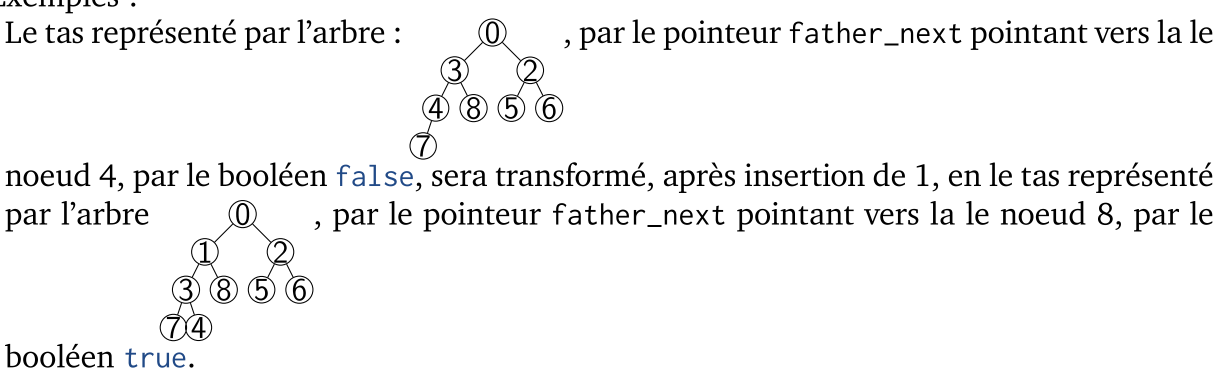
Exemples :

Dans l'arbre  le noeud suivant 4 est 5, dont le suivant est 6, dont le suivant est 7, dont le suivant est 8, dont le suivant est 4.

Q. 2 Définir la fonction `node prev(node cur)`, prenant en argument un noeud `cur` que l'on supposera être une feuille et retournant la prochaine feuille dans l'ordre inverse de la fonction précédente.

Q. 3 Définir la fonction `void insert(bheap* bt, int elem)` permettant l'insertion d'un élément dans un tas binomial.

Exemples :


Le tas représenté par l'arbre : , par le pointeur `father_next` pointant vers la le noeud 4, par le booléen `false`, sera transformé, après insertion de 1, en le tas représenté par l'arbre , par le pointeur `father_next` pointant vers la le noeud 8, par le booléen `true`.

Q. 4 Définir la fonction `void suppression(bheap* bt)` permettant la supression du minimum dans un `bheap`.

Exercice 3 : Le problème de Clément

On a vu en classe que la donnée d'un parcours préfixe, infixe, ou encore postfixe, ne définit pas uniquement un arbre.
On considère cependant maintenant la donnée de deux parcours :
— préfixe, infixe

♣. surtout, comme dans l'algorithme du cours

— postfixe, infixe
— préfixe, postfixe

Q. 1 Y-a-t'il alors unicité de l'arbre décrit dans certains cas ?

Q. 2 Dans les cas, s'il y en a, où la réponse est affirmative, donner une fonction OCaml permettant le calcul de l'arbre représenté.

Exercice 4 : Longueur de cheminement

On rappelle qu'un noeud peut être assimilé à son chemin (un mot de $\{0,1\}^\star$) depuis la racine d'un arbre. On note $|u|$ la longueur d'un mot u, ε le mot vide et $u \cdot v$ la concaténation de deux mots u et v. On note $\mathcal{N}_t$ l'ensemble des noeuds d'un arbre binaire t.
On définit la *longueur de cheminement* d'un arbre binaire t comme étant $\mathsf{long}(t) = \sum_{n \in \mathcal{N}_t} |n|$. On rappelle que la longueur du mot représentant le chemin de la racine d'un arbre à un noeud n est exactement la profondeur du noeud n. La longueur de cheminement d'un arbre est donc la somme des profondeurs de tous les noeuds.

Q. 1 Redéfinir, mais par induction, la fonction long. Votre définition pourra utiliser la fonction $s : \mathcal{B}_S \to \mathbb{N}$ donnant la taille d'un arbre binaire. Vous devez prouver, par induction, que votre redéfinition coïncide avec la fonction $\mathsf{long}(t)$.

Q. 2 Démontrer que $\forall t \in \mathcal{B}_S, \mathsf{long}(t) \leqslant \frac{\mathsf{s}(t)(\mathsf{s}(t)-1)}{2}$.

Q. 3 Prouver qu'à une taille fixée, un arbre minimisant la longueur de cheminement a ses feuilles placées sur deux niveaux adjacents.

Q. 4 Démontrer que $\forall t \in \mathcal{B}_S, \sum_{k=1}^{\mathsf{s}(t)} \lfloor \log_2(k) \rfloor \leqslant \mathsf{long}(t)$

Exercice 5 : Tri par tas

Q. 1 Implanter l'algorithme du tri par tas.