Structures de données en C

TD/TP 4 – Les arbres

Exercice 1 : Arbres binaires

1. Implanter la spécification ABin en utilisant les déclarations suivantes :

```
typedef ... arbre;
arbre empty();
int is_empty(const arbre);
arbre sad(arbre);
arbre sag(arbre);
```

2. Écrire les fonctions **taille** et **hauteur.**

On dit qu'un arbre est **filiforme** (*dégénéré*) si chaque nœud a au moins un fils vide :

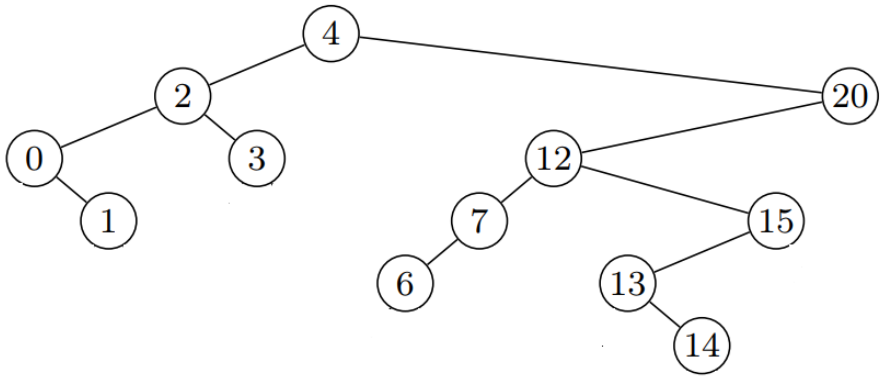
3. Dessiner tous les arbres filiformes de taille 1, 2 et 3.
4. Combien y a-t-il d'arbres filiformes de taille n ?
5. Écrire une fonction **est_filiforme**.
6. Montrer que pour tout arbre filiforme **t** on a l'égalité **taille(t) = hauteur(t)**

On dit qu'un arbre est un **peigne gauche** si le fils droit de chaque nœud est vide.
On définit de même les peignes droits.

7. Écrire une fonction **peigne(n)** qui retourne le peigne gauche de taille **n** donné.
8. Exécuter **est_filiforme(peigne(3))**. Qu'en pensez-vous ?

Exercice 2: Parcourt d'arbres binaires

1. Donner l'ordre de parcours des nœuds de l'arbre suivant, pour l'ordre préfixe. Quel est l'affichage si l'on affiche les étiquettes dans cet ordre ?



2. Même question pour les ordres infixe, et postfixe.
3. Même question pour un parcours en profondeur ?
4. Écrire un algorithme qui étant donné un arbre retourne la liste des étiquettes de cet arbre dans l'ordre de parcours préfixe.

Exercice 3 : Arbres binaires de recherches, suppression

Pour chacun des algorithmes suivants, on donnera la complexité.

1. Écrire une fonction qui étant donné un **ABR** non vide retourne la plus petite valeur de cet **ABR**.
2. Écrire une fonction qui étant donné un **ABR** non vide supprime et retourne la plus petite valeur de cet **ABR**.
3. Écrire une fonction qui étant donné un **ABR a** supprime la racine de cet **ABR**. Si **a** et les deux sous arbres gauche et droit sont tous non vide, on remplacera la racine par le plus petit nœud du sous arbre gauche de **a**. Pourquoi ce choix est-il correct ?
4. Écrire une fonction qui étant donné un élément ***e*** et un arbre **a** supprime **e** de **a** s'il est dans **a** et sinon laisse **a** inchangé.

Exercice 4 : Partitions d'arbres binaires de recherches

Écrire un programme qui étant données **Ar** un **ABR** et **e** un élément, retourne un **ABR Ar_e** qui ne contient que les éléments de **Ar** inférieur à la valeur à **e**.
Contrainte : L'algorithme ne doit parcourir qu'un seul chemin dans l'**ABR** sans comparer tous les nœuds à **e**.