

LO21 - TP N°4

Récurtivité & Arbre binaire

Exercice n°1

Considérons les trois suites d'entiers naturels $(u_n)_{n \geq 0}$, $(v_n)_{n \geq 0}$ et $(w_n)_{n \geq 0}$ définies par récurrence simultanée de la façon suivante :

$$\begin{aligned}u_0 &= 1, v_0 = 2, w_0 = 3, \\u_{n+1} &= 2.u_n + 3.v_n + w_n, \\v_{n+1} &= u_n + v_n + 2.w_n, \\w_{n+1} &= u_n + 4.v_n + w_n.\end{aligned}$$

Ecrire un programme en C récursif qui demande un entier n et affiche alors les valeurs de u_n , de v_n et de w_n .

Exercice n°2

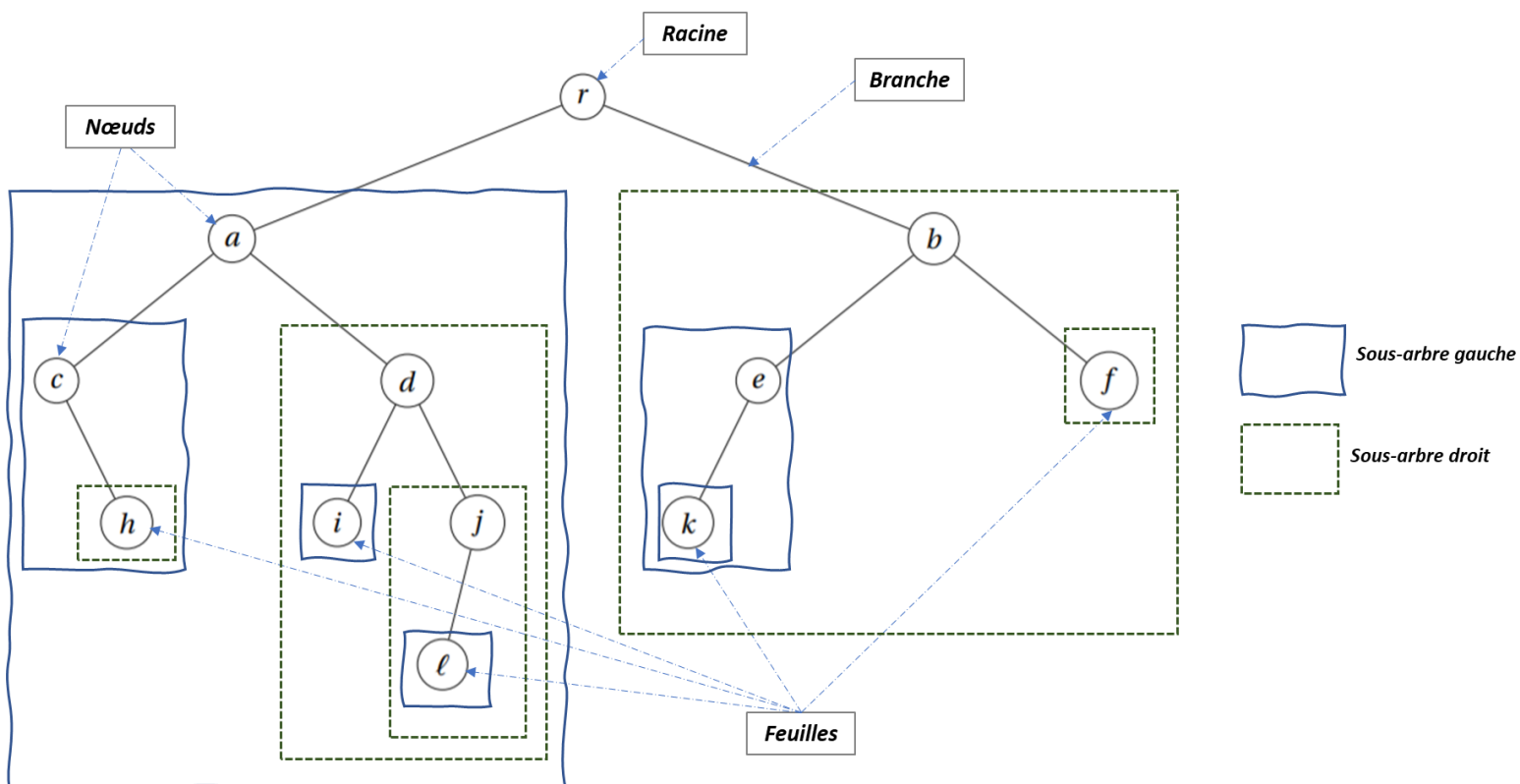
Définition d'un type abstrait de donnée « arbre binaire ».

Question 1.

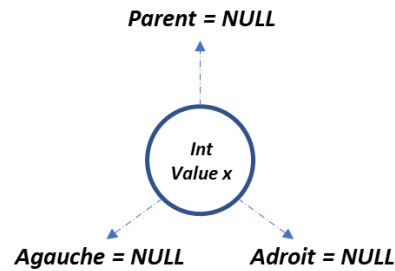
Écrire dans les fichiers « *arbre.h* » et « *arbre.c* » la définition de la structure d'un arbre binaire ainsi que les fonctions suivantes :

```
/* Structure d'un Arbre */
typedef struct Arbre
{
    int value;
    struct Arbre *Agauche;
    struct Arbre *Adroit;
    struct Arbre *parent;
}Arbre;
```

- Chaque nœud i de l'arbre est composé d'un pointeur vers le sous-arbre gauche de i , un pointeur vers le sous-arbre droit de i comme l'illustre la figure suivante :



- 1) Écrire une fonction ***cree_arbre()*** qui permet de créer un arbre. Il est nécessaire d'allouer l'espace nécessaire pour créer un arbre. On utilisera pour cela la fonction ***malloc***.



- 2) Écrire une fonction ***join_arbre()*** qui prend un entier ainsi que deux arbres et renvoie un arbre dont la racine ou le parent contient l'entier et les deux sous-arbres sont ceux donnés en paramètre.
- 3) Écrire une fonction (**réursive**) ***affiche_arbre_prefixe()*** qui affiche les valeurs des nœuds d'un arbre par ordre préfixe : **racine, fils gauche, fils droit**.
Exemple : l'affichage dans l'ordre préfixe de l'arbre de la figure 1 est : ***r, a, c, h, d, i, j, l, b, e, k, f***.
- 4) Écrire une fonction (**réursive**) ***affiche_arbre_postfixe()*** qui affiche les valeurs des nœuds d'un arbre par ordre postfixe : **fils gauche, fils droit, racine**.
Exemple : l'affichage dans l'ordre postfixe de l'arbre de la figure 1 est : ***h, c, i, l, j, d, a, k, e, f, b, r***.
- 5) Écrire une fonction (**réursive**) ***nombre_de_noeuds()*** qui calcule le nombre de nœuds (les feuilles et la racine sont incluses) d'un arbre binaire.
- 6) Écrire une fonction (**réursive**) ***vider_arbre()*** qui libère la mémoire occupée par tous les nœuds d'un arbre binaire.

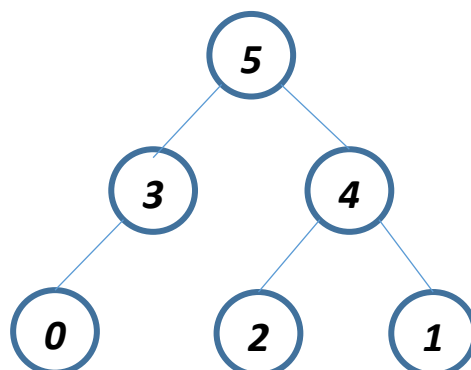
On utilisera donc les définitions de type suivantes :

```
/* Prototypes des fonctions */
```

```
Arbre *cree_arbre(int x);
Arbre *join_arbre(Arbre *gauche, Arbre *droit, int noeud);
void vider_arbre(Arbre *ar);
void affiche_arbre_prefixe(Arbre *ar);
void affiche_arbre_postfixe(Arbre *ar);
int nombre_de_noeuds (Arbre *ar);
```

Question 2.

Écrire dans un fichier « *main_arbre.c* » un programme qui permet de tester les fonctions de la question 1 en créant l'arbre suivant :



Exercice n°3

On introduit une phrase au clavier se terminant par un point (et ne contenant pas d'autre point que celui-ci). Le programme (**récuratif**) doit afficher cette phrase à l'écran dans l'ordre inverse. On utilisera pour cela les macro-instructions *putchar()* et *getchar()*.

Une session d'utilisation de ce programme sera, par exemple :

```
Ecrire une phrase :  
TP5 L021.  
Cette phrase dans l'ordre inverse  
est : .120L 5PT
```