Enoncé de l'exercice

On considère la structure ci-après pour implémenter un arbre binaire de recherche :

```
typedef struct node {
    unsigned int key;
    struct node *left;
    struct node *right;
} node;
```

1. Est-il avantageux d'utiliser exclusivement cette seule structure pour implementer un arbre binaire de recherche ? Justfiier votre choix.
2. On se propose dans la suite de faire usage de cette seule structure pour l'implementation de cet arbre. Répondez aux questions suivantes
    1. Ecrire une fonction permettant de rechercher la présence d'un nombre entier dans l'arbre;
    2. Ecrire une fonction servant à calculer la profondeur de l'arbre ;
    3. Ecrire une fonction permettant d'afficher les éléments stockés dans l'arbre de manière décroissante.

---

1. Est-il avantageux ?

Oui

2. Implémentation des fonctions

> Dans un arbre de rechcerche, les nombres superieurs à la raine sont placés a droite alors que les nombres inferierus a la racine sont placés a gauche

a. Fonction permettant de rechercher la présence d'un nombr entier dans l'arbre

```
// Ma version

int find(unsigned int number, node *tree) {
  if (!tree) // même chose que if (tree == NULL)
    return 0;

  if (tree->key == number)
    return 1;
```

```
  return find(number, tree->left) || find(number, tree->right);
}
```

Ou

```
// Version du professeur

int serchNoeud(node *tree, unsigned int key) {
    while(tree) {
        if (key == tree->key) return 1;

        if (key > tree->key) tree = tree -> right;
        else tree = tree -> left;
    }
    return 0;
}
```

Dans les deux fonctions, ont retourne 1 si le nombre est trouvé et 0 sinon

b. Fonction pour calculer la profondeur de l'arbre

Pour cette fonction nous aurons besoin d'une autre fonction définie comme suit :

```
int maximum(int a, int b){
    return a > b ? a : b;
}

 // cela equivaut à
// if (a > b)
//     return a;
// else
//     return b;
```

```
int depth(node *tree) {
  if (!tree) // Equivaut à if (tree == NULL)
    return 0;

  return (1 + maximum(depth(tree->left), depth(tree->right)));
}
```

c. Fonction pour afficher les éléments de l'arbre dans l'ordre décroissant

Comme expliqué plus haut, dans un arbre de recherche, les éléments supérierus à la racine sont placés a droite et les éléments infétieurs à la racine sont placés a gauche.

Si on veut afficher par ordre décroissant, il faut donc qu'on affiche les éléments a droite, ensuite la racine, et enfin les éléments a gauche

```
void printReverse(node *tree) {
  if (!tree) // Equivaut à if (tree == NULL)
    return;  // Ne rien faire et sortir de la fonction

  if (tree->right) // vérifie si l'enfant de droite n'est pas vide
    printReverse(tree->right);

  printf("%d ", tree->key);

  if (tree->left) // Vérifie si l'enfant de gauche n'est pas vide
    printReverse(tree->left);
}
```