## Consignes générales pour les TDs

FAITES BIEN LES SCHEMAS DEMANDES, **CE N’EST PAS OPTIONNEL** ET VOUS AIDERA À VISUALISER ET COMPRENDRE LES ALGORITHMES

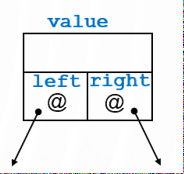
POUR LES EXERCICES COMPORTANT DE LA REDACTION DE CODE EN C, **UTILISEZ UN EDITEUR DE TEXTE OU UN IDE**, AFIN DE POUVOIR RECUPERER VOS TRAVAUX POUR LES TPs.

Les exercices précédés d’un symbole losange ◆ sont obligatoires.

# TD4 : Arbres binaires

## Partie 1) Visualiser et créer des arbres

Rappel : pour visualiser un arbre, vous pouvez utiliser l’une des deux formes de nœud suivantes :



Or u

Rappel : le type **t\_tree** est défini ainsi :

**typedef struct** s\_tree  
{  
 t\_node \*root;  
} t\_tree;

#### Exercice 1) Questions de cours

Question 1) définissez le type structure **t\_node** pour un nœud stockant des ‘char’ plutôt que des ‘int’

Question 2) Cela change-t-il la définition du type **t\_tree**?

Question 3) Écrivez le prototype puis la définition de la fonction **createEmptyTree()** qui retourne un arbre vide.

Question 4) Écrivez le prototype puis la définition de la fonction **createNode()** qui rend en paramètre une valeur de type ‘char’ et retourne un pointeur vers le nœud stockant cette valeur.

Question 5) Faites la visualisation des variables **mytree** et **p\_nouv** pour les étapes suivantes :

étape 1 :

**t\_tree mytree;**

**t\_node \*p\_nouv;**

étape 2:

**mytree=createEmptyTree();**

**p\_nouv = createNode(‘T’);**

étape 3:

**mytree.root = p\_nouv;**

étape 4:

**mytree.root->left = createNode(‘s’);**

étape 5:

**mytree.root->left->right = createNode(‘?’);**

#### Exercice 2) Création de sous arbres

Question 1) Écrivez la fonction **createTreeFromNode()** qui crée un arbre à partir d’un **t\_node \*** (ce **t\_node \*** devenant sa racine)

Question 2) Réécrire la fonction **createEmptyTree()** en utilisant **createTreeFromNode()**

Question 3) A partir de l’étape 5 de l’exercice 2, faites la visualisation des variables **mysubtree1** et **mysubtree2** d’après les instructions suivantes:

**t\_tree mysubtree1, mysubtree2;**

**mysubtree1 = createTreeFromNode(mytree.root->left)**

**mysubtree2 = createTreeFromNode(mysubtree1.root->left);**

## Partie 2) Algorithmes pour les arbres

#### Exercise 3) compter les noeuds

Pour cet exercice, la récursivité est la solution !

Question 1) Ecrivez le prototype de la fonction récursive **countNode()** : Cette fonction prend en paramètre une valeur de type **t\_node \***. Vous devriez trouver son type de retour d’après son nom.

Soit ‘pn’ le nom du paramètre de type **t\_node \***

Question 2) Quelle est la condition de fin de récursivité ?

Question 3) Si cette condition n’est pas vérifiée, alors on compte 1 nœud. Quels autres nœuds doit-on alors compter, à partir de celui où on se trouve ?

Question 4) Ecrivez la définition de la fonction **countNode()**

Question 5) Ecrivez la fonction **treeCountNode()**, qui compte le nombre de nœuds d’un arbre.

#### Exercise 4) Rechercher une valeur dans un arbre

Une fois de plus, la récursivité est de mise. On cherche à savoir si une valeur est stockée dans un arbre. Si oui, on veut récupérer le sous arbre dont la racine est le nœud qui stocke la valeur cherchée.

Étape 1) Trouver un **t\_node \*** dans un arbre

Question 1) Écrivez le prototype de la fonction récursive **seekValue()**. Cette fonction prend en paramètres :

La valeur à chercher (de type ‘char’)

Le **t\_node \*** à partir duquel on fait la recherche

Cette fonction retourne le pointeur (**t\_node \***) du nœud où se trouve la valeur cherchée

Soit **pn** le nom du paramètre de type **t\_node \***, et **val** le nom du paramètre de type ‘char’

* 1. Si la valeur cherchée n’est pas dans l’arbre, quelle valeur doit être retournée par cette fonction ?
  2. Quelle est la condition de fin de récursivité?

Question 2) Écrivez la définition de la fonction **seekValue()**

Question 3) Écrivez la fonction **treeSeekValue()** qui retourne le sous arbre dont la racine est le nœud trouvé par la fonction **seekValue()**

*Exercice 5)* Proposer une fonction récursive qui affiche toutes les valeurs stockées dans un arbre (on écrira d’abord une fonction récursive pour un **t\_node** **\***)