**Travail à faire (2) :**

* Implémenter un arbre binaire de recherche de mots.

On se donne un texte qui est une suite de phrases. Une phrase est une suite de mots qui sont séparés par des espaces. Ce texte on peut le saisir au clavier comme supposer qu’il est dans un fichier de texte (la plus conseillée). Donc, sûrement on aura des mots qui vont se répéter plusieurs fois…

Le but de l’application est par exemple d’afficher la liste triée des mots du texte en indiquant à côté de chaque mot le nombre de fois qu’il apparaît dans le texte.

Un arbre est : Une structure non linéaire qui permet d’obtenir des algorithmes plus performants que lorsqu’on utilise des structures de données linéaires

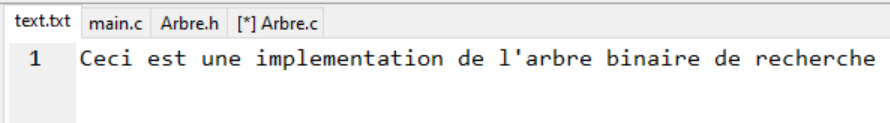
On peut l’utiliser pour représenter les mots d’un texte, pour ce faire **on va utiliser :**

Nœud, Elément, Booléen.

**Pour les opérations :**

* Arbre arbre\_vide(); pour créer arbre vide
* Bool est\_vide(); pour tester si l’arbre est vide ou non
* Arbre construire (Nœud \*r, Arbre gauche, Arbre droite) ; permet de construire un arbre
* Noeud racine (Arbre arbre); permet de retourner la racine d’une arbre
* Arbre droite (Arbre arbre); permet de retourner l’arbre droite d’une arbre
* Arbre gauche (Arbre arbre) ; permet de retourner l’arbre gauche d’une arbre
* Arbre insere (Arbre arbre, Element val) ; permet d’insérer un élément dans l’arbre
* void afficher (Arbre arbre) ; permet d’afficher l’arbre

On crée un fichier texte duquel on lie son contenu



Et puis on parcourt les mots, chaque mot parcouru sera ajouté à l’arbre.

S’il est inférieur au nœud racine on l’a ajoute dans fils gauche sinon on l’ajoute dans fils droite et puis on incrémente le nombre d’occurrence.

Et puis on affiche toute l’arbre, voici ce que ça donne :

void afficher (Arbre arbre) {

if (est\_vide(arbre)) {

return;

} else {

afficher (arbre->fils\_droite);

printf("%s : %d \n" , arbre->mot, arbre->occurence);

afficher (arbre->fils\_gauche);

}

}

