EX2 : Arbre lexicographique (corrigé)

PNoeud : pointeur sur Noeud
Le voir def : "typedef" sur le sujet.

①

```
PNoeud creer_noeud(char lettre){
    PNoeud pn = (PNoeud)malloc(sizeof(Noeud));
    if(pn==NULL){
        printf("Impossible allouer noeud\n");
        return NULL;
    }
    pn->lettre = lettre;        // valeurs par defaut
    pn->fin_de_mot = non_fin;
    pn->fils = NULL;
    pn->frere_suivant = NULL;
    return pn;
}
```

Figure 2 du TD :
liste frères : racine → [a] → [b] → NULL

② Pour chercher un mot dans l'arbre, on considère sa première lettre et on la cherche dans la liste des frères* de la racine. Si elle n'est pas dedans, alors le mot n'est pas dans l'arbre et on renvoie 0.

Si elle y est, alors 2 cas sont possibles : soit n le noeud associé à la lettre

1er cas : Si n->fin_de_mot = fin et que le mot ne contient qu'une seule lettre, alors on renvoie 1. (le mot existe dans l'arbre)

~~2ème cas : On applique la recherche au sous-arbre fils (l'arbre des lettres suivantes) et on~~

2ème cas : On cherche le reste du mot dans le sous-arbre fils n->fils ("arbre des lettres suivantes")

On va donc coder une fonction auxiliaire qui cherche le noeud correspondant à la première lettre du mot dans la liste des frères :

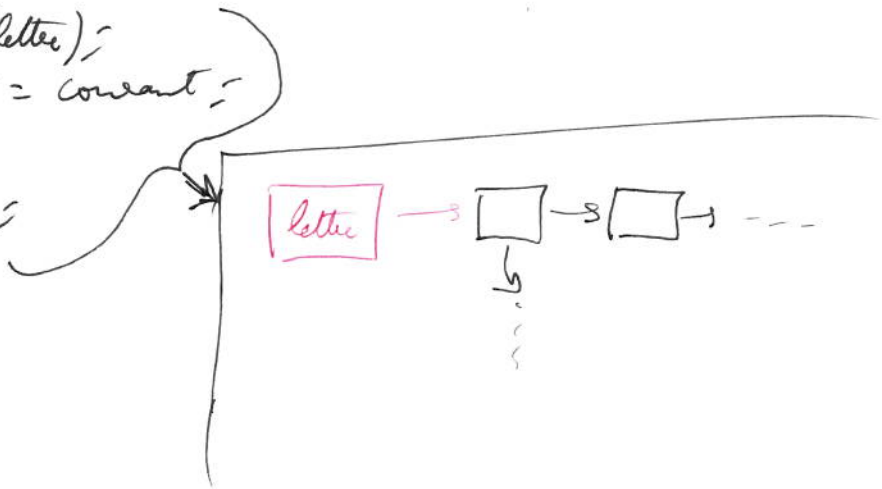
```
PNoeud chercher_lettre (PNoeud n, char lettre){
    if (n == NULL) return NULL; // liste vide => le noeud qu'on cherche n'existe pas.
    if (n->lettre == lettre) return n; // on a trouvé le noeud recherché
    if (n->lettre > lettre) return NULL; /* la liste est triée donc la lettre qu'on cherche aurait dû se trouver avant. */
    return chercher_lettre (n->frere_suivant, lettre); /* pas encore trouvé
                                                          => on cherche dans le reste de la liste. */
}

int rechercher_mot (PNoeud dico, char *mot){
    PNoeud n = chercher_lettre (dico, *mot) /* -> recherche première lettre dans la liste des frères de la racine (dico). */
    char *p_lettre_suivante = mot + 1; /* pointeur sur suite du mot, sur la lettre suivante. */
    if (n == NULL) return 0; /* la première lettre n'est pas là donc le mot ne figure pas dans le dico */
    if ((*p_lettre_suivante) == 0)
        return n->fin_de_mot == fin; /* Si la dernière lettre est une fin de mot dans l'arbre, alors le mot existe dans le dictionnaire */.
    return rechercher_mot (n->fils, p_lettre_suivante); /* Sinon, on continue la recherche des lettres suivantes dans le sous-arbre fils */
}
```

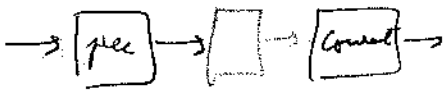
(3) Pour insérer un mot, les fonctions sont plus compliquées mais le raisonnement est le même.
Si on sait insérer une lettre dans la liste des frères, alors on peut, par récursivité, insérer un mot dans l'arbre.

On commence par l'insertion d'une lettre dans la liste des frères
modif => passe par pointeur

```
void inserer_lettre(PNoeud *racine, PNoeud *n_lettre, char lettre){
    PNoeud courant;
    PNoeud prec; // précède noeud courant dans la liste des frères
    prec = NULL;
    courant = *racine;
    if (courant == NULL){ // arbre vide
        *racine = creer_noeud(lettre);
        *n_lettre = *racine;
        return;
    }
    while (courant != NULL){ // Parcours de la liste des frères
        if (courant->lettre == lettre){ // la lettre existe déjà
            *n_lettre = courant;
            return;
        }
        if (courant->lettre > lettre){ // la lettre n'existe pas dans la liste et doit être placée avant le noeud courant.
            if (prec == NULL){ // le noeud courant est la racine
                prec = creer_noeud(lettre);
                prec->frere_suivant = courant;
                *racine = prec;
                *n_lettre = *racine;
            }
```



```
        else {
            *n_lettre = creer_noeud(lettre);
            (*n_lettre)->frere_suivant = courant;
            prec->frere_suivant = *n_lettre;
        }
        return;
    }
    // Si on est pas tombé dans un des cas précédent, on passe à l'élément
    // suivant dans la liste
    prec = courant;
    courant = courant->frere_suivant;
} //fin boucle
/* Si l'algo ne s'est pas encore arrêté, cela signifie que le lettre
n'existe pas dans la liste et doit être placée à la fin de cette même liste*/
*n_lettre = creer_noeud(lettre);
prec->frere_suivant = *n_lettre;
} //fin de la fonction.
```



On peut alors ajouter un mot récursivement :

```
PNoeud ajouter_mot(PNoeud racine, char *mot){
    char *p_lettre_suivante;
    PNoeud n = NULL;
    if (*mot == 0) return NULL;
    inserer_lettre(&racine, &n, *mot);
    p_lettre_suivante = mot+1;
    if ((*p_lettre_suivante) == 0)
        n->fin_de_mot = fin;
    else
        n->fils = ajouter_mot(n->fils, p_lettre_suivante);
    return racine;
}
```

Nombre de comparaisons dans le pire des cas : $26 \times m$

m : longueur moyenne d'un mot.

⇒ très faible par rapport à un dictionnaire stocké sous forme de liste.

Occupation mémoire :

Nœud : lettre : 1o
entier (int) : 4o
pointeurs : 2×8o en 64 bits
} 21o

Dans le pire des cas (quasi impossible) aucune lettre n'est partagée parmi les mots du dictionnaire :

$$21 \times N \times m = 21 \times 130\,000 \times 5 = 13 Mo$$

Dans la pratique (TME) : 5,2 Mo. > 3,7 Mo (ABR)

	ABR	arbre lexicographique
occupation mémoire	3,7 Mo	5,2 Mo
complexité	$O(\log_2(N))$	$O(m)$

⇒ Compromis entre efficacité calculatoire et occupation mémoire.