Similitude dans l'implémentation

Liste :

```c
typedef struct Maillon Maillon;
typedef struct Liste Liste;

struct Maillon {
    int valeur;
    Maillon *suivant;
};


struct Liste {
    Maillon *racine;
};
```

Arbre :

```c
typedef struct Noeud Noeud;
typedef struct Arbre Arbre;

struct Noeud {
    int cle;
    Noeud *gauche;
    Noeud *droit;
};

struct Arbre {
    Noeud *racine;
};
```

```c
/* Fonction 'helper' pour liberer la memoire allouee aux noeuds d'un arbre */
void free_arbre_helper(Noeud *noeud) {
	if (noeud != NULL) {
		free_arbre_helper(noeud->gauche);
		free_arbre_helper(noeud->droit);
		free(noeud);
	}
}

/* Libere la memoire allouee a un arbre et a tous ses noeuds */
void free_arbre(Arbre *A) {
	if (A) {
		if (A->racine != NULL) {
			free_arbre_helper(A->racine);
		}
		free(A);
	}
}
```

```c
/* Fonction "helper" pour le parcours prefixe */
void parcours_prefixe_helper(Noeud *mon_noeud) {
	if (mon_noeud != NULL) {
		printf("%d ", mon_noeud->cle);
		parcours_prefixe_helper(mon_noeud->gauche);
		parcours_prefixe_helper(mon_noeud->droit);
	}
}

/* Parcours prefixe d'un arbre binaire */
void parcours_prefixe(Arbre *mon_arbre) {
	if (mon_arbre == NULL) return;
	parcours_prefixe_helper(mon_arbre->racine);
}
```

```
/* Fonction "helper" pour compter le nombre de noeuds d'un arbre binaire */
int nombre_noeuds_helper(Noeud *mon_noeud) {
        if (mon_noeud == NULL) return 0;
        return 1 + nombre_noeuds_helper(mon_noeud->gauche)
                 + nombre_noeuds_helper(mon_noeud->droit);
}
```

-> récursivité dans le return pour compter le nombre de noeuds

```
/* Fonction "helper" pour la calcul de la profondeur d'un arbre binaire. */
int profondeur_helper(Noeud *mon_noeud) {
        int profondeur_g, profondeur_d;

        if (mon_noeud == NULL) return 0;

        profondeur_g = profondeur_helper(mon_noeud->gauche);
        profondeur_d = profondeur_helper(mon_noeud->droit);

        return (profondeur_g < profondeur_d ? 1 + profondeur_d : 1 + profondeur_g);
}
```

-> si profondeur_g < profondeur_d,
alors 1 + profondeur_d,
sinon 1 + profondeur_g

```
bool est_abr_helper(Noeud *mon_noeud, int cle_min, int cle_max) {
        if (mon_noeud == NULL) return true;
        if (mon_noeud->cle < cle_min || mon_noeud->cle > cle_max) return false;
        return est_abr_helper(mon_noeud->gauche, cle_min, mon_noeud->cle)
                && est_abr_helper(mon_noeud->droit, mon_noeud->cle + 1, cle_max);
}

/* Renvoie true si l'arbre binaire transmis en argument est un arbre binaire de
 * recherche, et false sinon. */
bool est_abr(Arbre *mon_arbre) {
        if (mon_arbre == NULL || mon_arbre->racine == NULL) return false;
        return est_abr_helper(mon_arbre->racine, INT_MIN, INT_MAX);
}
```

-> **INT_MIN**, **INT_MAX** sont des constantes définies dans limits.h
donc **#include <limits.h>**