```c
/*****************************************/
/*** Gestion d'arbres binaires dynamiques ***/
/*****************************************/
#include <stdio.h>
#include <stdlib.h>

#define MALLOC(x)((x * ) malloc(sizeof(x)))
#define TRUE 1
#define FALSE 0
#define ARBRENULL NULL

/*
 * Definition du type d'Element
 */
typedef int Element;

#define ELEMENTNULL -1

/*
 * Structure Noeud et Arbre = pointeur sur Noeud
 */
typedef struct arb {
  Element elmt;
  struct arb* fg;
  struct arb* fd;
} Arbre;

typedef Arbre* PArbre;

/*
 * Structure de file d'arbre basee sur la liste
 */
typedef struct lst {
       PArbre val;
       struct lst * suiv;
} Liste;
typedef Liste* Pliste;

typedef struct file {
       Pliste tete;
       Pliste queue;
} File;


/*********************************
 * Prototypes
 */
Pliste creerEltListe(PArbre, Pliste);
void creerFile(File *);
void enfiler(File *, PArbre);
PArbre defiler(File *);

PArbre creerArbre(Element );
int estVide(PArbre );
int estFeuille(PArbre );
Element racine(PArbre );
PArbre modifierRacine(PArbre , Element );
int existeFilsGauche(PArbre );
```

```c
int existeFilsDroit(PArbre );
PArbre filsGauche(PArbre );
PArbre filsDroit(PArbre );
PArbre ajouterFilsGauche(PArbre , Element );
PArbre ajouterFilsDroit(PArbre , Element );
PArbre supprimerFilsGauche(PArbre a);
PArbre supprimerFilsDroit(PArbre );
void traiter(Element );
void parcoursPrefixe(PArbre );
void parcoursInfixe(PArbre );
void parcoursPostfixe(PArbre );
void parcoursLargeur(PArbre );
int max(int , int );
int hauteur(PArbre );
int taille(PArbre );
int nbFeuilles(PArbre );
void affArbre(PArbre );


/*************************
 * Fonctions sur les files
 */
/*
 * Creation d'un noeud de valeur x et de suivant suiv
 * retourne le noeud cree
 */
Pliste creerEltListe(PArbre x, Pliste suiv) {
	Pliste ptr;
	if ((ptr = MALLOC(Liste)) == NULL) {
		fprintf(stderr, "ERREUR ALLOCATION MEMOIRE");
		exit(1);
	}
	ptr->val = x;
	ptr->suiv = suiv;
	return ptr;
}

/*
 * Initialisation des tete et queue de la file
 */
void creerFile(File *f) {
	f->tete = NULL;
	f->queue = NULL;
}

/*
 * Enfilement : on insere l'element
 * tete et queue sont modifiees donc passees par adresse (PrtListe *)
 */
void enfiler(File *f, PArbre x) {
	Pliste ptr = creerEltListe(x, NULL);
	if (f->queue) f->queue->suiv = ptr;	// queue pointe sur le nouvel element
	else f->tete = ptr;				// si file vide = nouvel element
devient la tete
	f->queue = ptr;					// nouvel element =
nouvelle queue
}
```

```c
/*
 * Defilement : tete et queue sont modifiees donc passees par adresse (PrtListe *)
 */
PArbre defiler(File *f) {
	Pliste ptr = f->tete;
	PArbre x;
	if (!ptr) return ARBRENULL;		// File vide
	f->tete = ptr->suiv;		// tete passe au suivant
	if (f->queue == ptr) f->queue = NULL; // File devient NULL tete = queue
	x = ptr->val;			// on recupere la valeur
	free(ptr);			// on libere l'element
	return x;
}


/*********************************
 * Fonctions sur les arbres
 */
PArbre creerArbre(Element e) {
	PArbre a;
	if ((a = MALLOC(Arbre)) == NULL) {
		fprintf(stderr, "ERREUR ALLOCATION MEMOIRE");
		exit(1);
	}
	a->elmt = e;
	a->fg = NULL;
	a->fd = NULL;
	return a;
}

int estVide(PArbre a) {
  return (a == ARBRENULL);
}

int estFeuille(PArbre a) {
	return !(a->fg || a->fd);
}

Element racine(PArbre a) {
	return (a ? a->elmt : ELEMENTNULL);
}

PArbre modifierRacine(PArbre a, Element e) {
	if (a)
		a->elmt = e;
	return a;
}

int existeFilsGauche(PArbre a) {
	return (a->fg ? TRUE : FALSE);
}

int existeFilsDroit(PArbre a) {
	return (a->fd ? TRUE : FALSE);
}

PArbre filsGauche(PArbre a) {
	return a->fg;
```

```
}

PArbre filsDroit(PArbre a) {
	return a->fd;
}

PArbre ajouterFilsGauche(PArbre a, Element e) {
	if (! existeFilsGauche(a))
		a->fg = creerArbre(e);
	return a;
}

PArbre ajouterFilsDroit(PArbre a, Element e) {
	if (! existeFilsDroit(a))
		a->fd = creerArbre(e);
	return a;
}

PArbre supprimerFilsGauche(PArbre a) {
	if (existeFilsGauche(a)) {
		if (existeFilsGauche(a->fg))
			supprimerFilsGauche(a->fg);
		if (existeFilsDroit(a->fg))
			supprimerFilsDroit(a->fg);
		free(a->fg);
		a->fg = NULL;
	}
	return a;
}

PArbre supprimerFilsDroit(PArbre a) {
	if (existeFilsDroit(a)) {
		if (existeFilsGauche(a->fd))
			supprimerFilsGauche(a->fd);
		if (existeFilsDroit(a->fd))
			supprimerFilsDroit(a->fd);
		free(a->fd);
		a->fd = NULL;
	}
	return a;
}

void traiter(Element e) {
	printf("%d ", e);
}

void parcoursPrefixe(PArbre a) {
	if (! estVide(a)) {
		traiter(a->elmt);
		parcoursPrefixe(a->fg);
		parcoursPrefixe(a->fd);
	}
}

void parcoursInfixe(PArbre a) {
	if (! estVide(a)) {
		parcoursInfixe(a->fg);
		traiter(a->elmt);
```

```
		parcoursInfixe(a->fd);
	}
}

void parcoursPostfixe(PArbre a) {
	if (! estVide(a)) {
		parcoursPostfixe(a->fg);
		parcoursPostfixe(a->fd);
		traiter(a->elmt);
	}
}

void parcoursLargeur(PArbre a) {
	PArbre noeud;
	File f;
	if (! estVide(a)) {
		creerFile(&f);
		enfiler(&f, a);
		while ((noeud = defiler(&f))) {
			traiter(noeud->elmt);
			if (existeFilsGauche(noeud)) enfiler(&f, filsGauche(noeud));
			if (existeFilsDroit(noeud)) enfiler(&f, filsDroit(noeud));
		}
	}
}

/*
 * Fonctions utiles sur les arbres
 */
int max(int a, int b) {
	return (a<b ? b : a);
}

int hauteur(PArbre a) {
	if (estVide(a)) return -1;
	if (estFeuille(a)) return 0;
	return 1 + max(hauteur(filsGauche(a)), hauteur(filsDroit(a)));
}

int taille(PArbre a) {
	if (estVide(a) || estFeuille(a)) return 0;
	return 1 + taille(filsGauche(a)) + taille(filsDroit(a));
}

int nbFeuilles(PArbre a) {
	if (estVide(a)) return 0;
	if (estFeuille(a)) return 1;
	return nbFeuilles(filsGauche(a)) + nbFeuilles(filsDroit(a));
}

void affArbre(PArbre a) {
	printf("Hauteur = %d\n", hauteur(a));
	printf("Taille = %d\n", taille(a));
	printf("nbFeuilles = %d\n", nbFeuilles(a));
	if (estVide(a)) {
		puts("Arbre vide");
	} else {
		printf("Parcours prefixe  : "); parcoursPrefixe(a); puts("");
```

```
                printf("Parcours infixe   : "); parcoursInfixe(a); puts("");
                printf("Parcours postfixe : "); parcoursPostfixe(a); puts("");
                printf("Parcours largeur  : "); parcoursLargeur(a); puts("");
        }
        puts("");
}

int main(){
        PArbre a = NULL;
        PArbre p = NULL;

        affArbre(a);
        a = creerArbre(1);
        a = ajouterFilsGauche(a, 2);
        a = ajouterFilsDroit(a, 8);
        p = filsGauche(a);
        p = ajouterFilsGauche(p, 3);
        p = ajouterFilsDroit(p, 6);
        p = filsGauche(p);
        p = ajouterFilsGauche(p, 4);
        p = ajouterFilsDroit(p, 5);
        p = filsDroit(filsGauche(a));
        p = ajouterFilsDroit(p, 7);
        p = filsDroit(a);
        p = ajouterFilsGauche(p, 9);
        p = ajouterFilsDroit(p, 10);
        affArbre(a);

        // supprimerFilsGauche(a);
        supprimerFilsGauche(filsGauche(filsGauche(a)));
        supprimerFilsGauche(filsGauche(filsGauche(a)));
        supprimerFilsDroit(filsGauche(filsGauche(a)));
        supprimerFilsDroit(filsGauche(filsGauche(a)));
        affArbre(a);

        supprimerFilsGauche(a);
        affArbre(a);
        supprimerFilsDroit(a);
        affArbre(a);

        return 0;
}
```