```
/*
 ------------------------------------------------------------------------------
 Nom du fichier : listes_dynamiques.c
 Auteur(s)      : Romain Fleury, Victor Nicolet
 Date creation  : 26.04.2023

 Description    : Librairie permettant la gestion de listes doublement chaînées
                  non circulaires

 Remarque(s)    : le status "POSITION_NON_VALIDE" n'est jamais utilisé car la seule
                  fonction qui utilise "position" ne retourne rien (void)

 Compilateur    : Mingw-w64 gcc 12.2.0
 ------------------------------------------------------------------------------
*/
#include <stdlib.h>
#include <stdio.h>
#include "listes_dynamiques.h"

Liste *initialiser(void) {
    Liste *lptr = (Liste *) calloc(1, sizeof(Liste));

    return lptr;
}

bool estVide(const Liste *liste) {
    if (liste->tete == NULL && liste->queue == NULL) {
        return true;
    }

    return false;
}

size_t longueur(const Liste *liste) {
    if (estVide(liste)) {
        return 0;
    } else {
        Element *e = liste->tete;
        if (e->suivant == NULL) {
            return 1;
        } else {
            size_t compteur = 1;
            while (e->suivant != NULL) {
                ++compteur;
                e = e->suivant;
            }
            return compteur;
        }
    }
}

void afficher(const Liste *liste, Mode mode) {
    if (estVide(liste)) {
        printf("[]");
    } else {
        if (mode) { // forward
            Element *eptr = liste->queue;
            printf("[");
            for (size_t i = 0; i < longueur(liste); i++) {
                printf("%d", eptr->info);
                if (i != longueur(liste) - 1) { printf(","); }
                eptr = eptr->precedent;
            }
            printf("]");
        } else { // backward
            Element *eptr = liste->tete;
            printf("[");
            for (size_t i = 0; i < longueur(liste); i++) {
                printf("%d", eptr->info);
                if (i != longueur(liste) - 1) { printf(","); }
                eptr = eptr->suivant;
            }
            printf("]");
        }
    }
}
```

```
Status insererEnTete(Liste *liste, const Info *info) {
    Element *eptr = (Element *) calloc(1, sizeof(Element));
    if (eptr != NULL) {
        if (estVide(liste)) {
            liste->tete = eptr;
            liste->queue = eptr;
            liste->tete->info = *info;

            return OK;
        } else {
            Element *tmp = liste->tete;
            liste->tete = eptr;
            liste->tete->suivant = tmp;
            liste->tete->suivant->precedent = liste->tete;
            liste->tete->info = *info;

            return OK;
        }
    } else {
        return MEMOIRE_INSUFFISANTE;
    }
}

Status insererEnQueue(Liste *liste, const Info *info) {
    Element *eptr = (Element *) calloc(1, sizeof(Element));
    if (eptr != NULL) {
        if (estVide(liste)) {
            liste->tete = eptr;
            liste->queue = eptr;
            liste->queue->info = *info;

            return OK;
        } else {
            Element *tmp = liste->queue;
            liste->queue = eptr;
            liste->queue->precedent = tmp;
            liste->queue->precedent->suivant = liste->queue;
            liste->queue->info = *info;

            return OK;
        }
    } else {
        return MEMOIRE_INSUFFISANTE;
    }
}

Status supprimerEnTete(Liste *liste, Info *info) {
    if (estVide(liste)) {
        return LISTE_VIDE;
    } else {
        if (longueur(liste) == 1) {
            free(liste->tete);
            liste->tete = NULL;
            liste->queue = NULL;

            return OK;
        } else {
            Element *eptr = liste->tete;
            *info = eptr->info;
            liste->tete = liste->tete->suivant;
            free(eptr);
            liste->tete->precedent = NULL;

            return OK;
        }
    }
}
```

```
Status supprimerEnQueue(Liste *liste, Info *info) {
    if (estVide(liste)) {
        return LISTE_VIDE;
    } else {
        if (longueur(liste) == 1) {
            free(liste->queue);
            liste->tete = NULL;
            liste->queue = NULL;

            return OK;
        } else {
            Element *eptr = liste->queue;
            *info = liste->queue->info;
            liste->queue = liste->queue->precedent;
            free(eptr);
            liste->queue->suivant = NULL;

            return OK;
        }
    }
}

void supprimerSelonCritere(Liste *liste,
                                    bool (*critere)(size_t position, const Info *info)) {
    if (estVide(liste)) {
        return;
    } else {
        Element *eptr = liste->tete;
        size_t i = 0; // position physique dans la liste
        size_t p = 0; // position relative dans la liste (paramètres de la fonction critère)
        Info x;
        while (eptr != NULL) {
            if (critere(p, &eptr->info)) {
                if (i == 0) { // effacer le premier
                    supprimerEnTete(liste, &x);
                } else if (i == longueur(liste)) { // effacer le dernier
                    supprimerEnQueue(liste, &x);
                    break; // superflu ?
                } else { // Effacer au milieu
                    Element *tmp = eptr; // avancer dans la liste jusqu'a l'élément à supprimer
                    eptr->suivant->precedent = eptr->precedent;
                    eptr->precedent->suivant = eptr->suivant;
                    free(tmp);
                    tmp = NULL;
                }
                eptr = liste->tete;
                for (size_t j = 1; j < i; j++) { eptr = eptr->suivant; }
            } else {
                i++;
                eptr = eptr->suivant;
            }
            ++p;
        }
    }
}

void vider(Liste *liste, size_t position) {
    if (position >= longueur(liste)) {
        return;
    } else {
        Element *eptr = liste->tete;
        Info x;

        for (size_t i = 0; i < position; i++)
            eptr = eptr->suivant;
        while (liste->queue != eptr->precedent)
            supprimerEnQueue(liste, &x);
    }
}
```

```
bool sontEgales(const Liste *liste1, const Liste *liste2) {
    if (longueur(liste1) == longueur(liste2)) {
        Element *ptr1 = liste1->tete, *ptr2 = liste2->tete;
        for (size_t i = 0; i < longueur(liste1); i++) {
            if (ptr1->info != ptr2->info) {
                return false;
            }
        }
        return true;
    } else {
        return false;
    }
}
```