Rapport Personnel - Projet Liste Chaînée

Contexte du Projet

Ce rapport détaille mon travail sur l'implémentation d'un programme en langage C permettant de gérer des listes chaînées, avec une fonctionnalité principale de suppression de toutes les occurrences d'un élément donné.

Travail Réalisé

1. Implémentation des Fonctions Principales

J'ai développé plusieurs fonctions essentielles pour la gestion des listes chaînées :

- Fonction `creer()` : Permet la création dynamique d'une liste chaînée à partir des entrées utilisateur
- Fonction `afficher()` : Affiche la liste de manière lisible avec des flèches indiquant les liens
- Fonction `supprimerOccurrences()` : Supprime toutes les occurrences d'une valeur spécifique dans la liste
- Fonction `supprimerListe()` : Libère la mémoire allouée pour éviter les fuites mémoires

2. Difficultés Rencontrées

Au cours du développement, j'ai rencontré des difficultés significatives lors de l'implémentation de la fonction `creer()`, particulièrement dans l'établissement du lien entre les cellules de la liste.

La gestion des pointeurs `tete` et `queue` ainsi que la liaison correcte entre les nœuds successifs m'ont demandé une attention particulière et plusieurs tentatives avant d'obtenir une structure cohérente. J'ai dû bien comprendre comment maintenir la référence au dernier élément tout en conservant l'accès à la tête de la liste.

3. Coordination et Organisation

En plus de mon travail de programmation, j'ai pris en charge **l'organisation et la gestion du dossier Algorithmique** contenant tous les programmes développés par le groupe. Cette responsabilité incluait :

- La compilation et l'organisation de tous les codes sources
- La vérification de la cohérence des fichiers
- La structuration du dossier pour faciliter l'accès aux différents programmes
- La coordination avec les autres membres du groupe

Résultats

Le programme final fonctionne correctement et permet :
- La création interactive de listes chaînées
- L'affichage clair de la structure de données
- La suppression efficace de toutes les occurrences d'un élément
- Une gestion appropriée de la mémoire

Compétences Développées

Ce projet m'a permis de renforcer mes compétences en :
- Manipulation de structures de données dynamiques
- Gestion de la mémoire en C (allocation et libération)
- Débogage de code impliquant des pointeurs
- Organisation et gestion de projet collaboratif

Code Source Complet

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Liste{
    int elt;
    struct Liste* suiv;
}*liste;

// Fonction pour créer la liste
liste creer(){
    int m;
    printf("Entrer le nombre d'elements de la liste: ");
    scanf("%d", &m);

    liste tete = NULL;
    liste queue = NULL;

    for(int i = 0; i < m; i++){
        int valeur;
        printf("Entrer l'element %d: ", i+1);
        scanf("%d", &valeur);

        liste noeud = (liste)malloc(sizeof(struct Liste));
        if (noeud == NULL){
            printf("Erreur d'allocation\n");
            exit(1);
        }

        noeud->elt = valeur;
        noeud->suiv = NULL;

        if(tete == NULL){
            tete = noeud;

            queue = noeud;
        } else {
            queue->suiv = noeud;
            queue = noeud;
        }
    }
    return tete;
}
```

```
// Fonction pour afficher la liste
void afficher(liste l){
    if(l == NULL){
        printf("Liste vide\n");
        return;
    }

    liste p = l;
    while(p != NULL){
        printf("%d", p->elt);
        if(p->suiv != NULL){
            printf(" -> ");
        }
        p = p->suiv;
    }
    printf(" -> NULL\n");
}

// Fonction pour supprimer toutes les occurrences d'une valeur
liste supprimerOccurrences(liste l, int val){
    // Supprimer les occurrences en début de liste
    while(l != NULL && l->elt == val){
        liste temp = l;
        l = l->suiv;
        free(temp);
    }

    // Si la liste est vide après suppression
    if(l == NULL){
         return NULL;
    }

    // Supprimer les occurrences dans le reste de la liste
    liste courant = l;
    while(courant->suiv != NULL){
        if(courant->suiv->elt == val){
            liste temp = courant->suiv;
            courant->suiv = temp->suiv;
            free(temp);
        } else {
            courant = courant->suiv;
        }
    }
    return l;
}

// Fonction pour supprimer toute la liste (libérer la mémoire)
void supprimerListe(liste l){
    liste q = l;
    while(q != NULL){
        liste temp = q;
        q = q->suiv;
```

```
        free(temp);
    }
}

int main(){
    liste l = NULL;

    // Créer la liste
    l = creer();

    if(l == NULL){
        printf("Aucun element dans la liste\n");
        return 0;
    }

    // Afficher la liste initiale
    printf("\nListe initiale: ");
    afficher(l);

    // Demander la valeur à supprimer
    int val;
    printf("\nEntrer l'element a supprimer: ");
    scanf("%d", &val);

    // Supprimer toutes les occurrences
    l = supprimerOccurrences(l, val);

    // Afficher la liste après suppression
    printf("\nListe apres suppression: ");
    afficher(l);

    // Libérer la mémoire
    supprimerListe(l);

    return 0;
}
```

Conclusion

Malgré les difficultés initiales, notamment sur la liaison des cellules, j'ai réussi à implémenter un programme fonctionnel. La prise en charge du dossier Algorithmique m'a également permis de développer mes compétences organisationnelles et ma capacité à coordonner le travail d'équipe.