Projet de fin de module.

Gestion Bancaires

Encadrer par : Pr. Omari Kamal

Réaliser par : Malka Fatima

Medhousse Aya

Mahraja Fatima-zahra

Date de présentation : Le 19 mai 2025.

Année universitaire 2024-2025

Remerciement

Tout d’abord, nous remercions Allah, le Tout-Puissant, de nous avoir guidés et permis de réaliser ce travail.

Nous adressons nos sincères remerciements à Monsieur Omari Kamal pour l’éducation qu’il nous a transmise et les informations qu’il nous a fournies tout au long de ce projet.

Nous remercions également les outils et plateformes de programmation qui nous ont permis de mener à bien ce projet de gestion bancaire et d’approfondir nos compétences.

Un grand merci à nos camarades pour leur aide, leur soutien et leurs encouragements. Leur présence a été précieuse durant cette expérience.

Enfin, nous remercions nos familles pour leur soutien constant, leur patience et leurs encouragements.

À toutes ces personnes, nous disons un grand merci du fond du cœur.

Résumer

Ce mini-projet, réalisé par un groupe de trois étudiantes, consiste à développer une application en langage C dédiée à la gestion des virements bancaires, en exploitant des structures de données telles que les listes chaînées, les piles et les files.

Le programme offre un menu interactif qui permet à l'utilisateur de gérer les virements de façon dynamique : ajout, suppression, modification, recherche, affichage et exécution.

Ce travail nous a permis d'appliquer les concepts théoriques appris en cours à un problème concret lié à la gestion bancaire.

Abstract

This mini-project, carried out by a group of three students, consists of developing a C language application dedicated to managing bank transfers using data structures such as linked lists, stacks, and queues.

The program offers an interactive menu that allows the user to dynamically manage transfers: adding, deleting, modifying, searching, displaying, and executing them. This work enabled us to apply theoretical concepts learned in class to a real-world problem related to banking operations.

This experience helped us improve our C programming skills and strengthen our teamwork through effective task distribution and collaboration.

Table des matières :

Objectifs et Contexte du Mini-Projet

Dans le cadre de notre formation, nous avons réalisé un mini-projet en langage C portant sur **la gestion des virements bancaires**. Ce projet s'inscrit dans une logique d'apprentissage pratique des structures de données dynamiques, telles que les **listes chaînées**, les **piles** et les **files**, tout en répondant à un besoin concret : la gestion informatique de transactions bancaires.

Avec la digitalisation croissante du secteur bancaire, il devient essentiel de comprendre comment structurer, stocker et manipuler des données liées aux opérations financières.

Ce projet nous a permis de concevoir un programme capable de gérer une série de virements bancaires, en intégrant les fonctionnalités suivantes :

- ajout d'un virement (au début, à la fin, ou après un virement spécifique),
- modification ou suppression d'un virement,
- recherche avancée (par montant, compte source ou compte bénéficiaire),
- affichage de tous les virements enregistrés,
- et enfin l'exécution d'un virement à travers une structure adaptée (pile ou file).

Ce mini-projet a pour objectif principal de consolider nos acquis en programmation C, en particulier la gestion dynamique de la mémoire et l'organisation du code en modules clairs et réutilisables. Travailler en groupe nous a également permis de développer des compétences en communication, en répartition des tâches, et en travail collaboratif, ce qui est indispensable pour nos projets futurs.

Méthodologie

Pour la réalisation de ce projet de gestion de virements bancaires, nous avons adopté une méthodologie progressive, adaptée à notre niveau d'apprentissage et à nos ressources disponibles en tant qu'étudiantes de deuxième année. Le projet a été mené en plusieurs étapes :

1. **Analyse du sujet et définition des besoins**

Nous avons commencé par une réflexion commune pour comprendre les fonctionnalités que notre application devait proposer, comme l'ajout, la suppression, la modification, la recherche et l'exécution de virements bancaires. Cette étape nous a permis de définir les principales structures de données à utiliser, notamment les listes chaînées, les piles et les files.

2. **Répartition des tâches au sein du groupe**

Le travail a été réalisé en groupe, en suivant une progression collective :

- On a collaboré toutes les trois à la mise en place des structures de données (pile, file, liste chaînée).
- On a ensuite conçu ensemble l'interface du menu principal et toute la logique d'exécution.
- On a développé en groupe les différentes opérations de gestion des virements (ajout, suppression, modification, etc.).
- On a pris soin de terminer chaque étape avant de passer à la suivante, tout en testant régulièrement le fonctionnement du programme.
- On a aussi rédigé le rapport et préparé la présentation ensemble, en partageant les idées et les explications.

3. Recherche et documentation

Pour combler certaines lacunes et approfondir notre compréhension, nous avons mené des recherches à l'aide de plusieurs outils et ressources :

- Visual Studio Code comme environnement de développement,
- Un compilateur C (GCC) pour tester et corriger le code,
- GitHub pour sauvegarder notre travail et suivre l'évolution du projet.
- Des tutoriels sur YouTube,
- Des sites éducatifs comme GitHub , Codecademy..
- ChatGPT pour obtenir des explications rapides et claires,
- Et bien sûr, nos cours et supports pédagogiques.

4. Développement progressif du code

Nous avons commencé par la mise en place des structures de données (liste chaînée, pile, file), puis nous avons développé les fonctions de base : ajout, suppression, modification, recherche, etc. Chaque étape a été testée pour assurer la cohérence et le bon fonctionnement du programme.

5. Réunions de groupe et discussions régulières

Nous avons tenu des réunions régulières pour faire le point sur l'avancement, partager nos difficultés, proposer des solutions ensemble, et assurer une cohérence dans le code.

6. Préparation du rapport et de la présentation

À la fin du projet, nous avons entamé la rédaction du rapport technique en détaillant chaque partie réalisée, puis nous avons préparé une présentation pour partager notre démarche, nos choix techniques et les compétences acquises.

Réalisation du code et Résultats

Code principale

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

typedef struct virement {
    float montant;
    char compte_source[100];
    char compte_beneficiaire[100];
    char date[100];
    int id;
    struct virement* next;
    struct virement* prev;
} virement;

typedef struct pile {
    virement data;
    struct pile* next;
} pile;

typedef struct file {
    virement data;
    struct file* next;
} file;

virement* head = NULL;
pile* sommet = NULL;
file* debut = NULL;
file* fin = NULL;
```

/* on trouve ici les structures principales d'un virement avec les infos (montant, client, ID...)

/* implémentation d'une pile pour enregistrer les virements supprimés

/* implémentation d'une file pour stocker les virements exécutés

/*Ajout d'un virement au début de la liste chaînée. Utilisation de l'allocation dynamique.

```c
void ajouterViremmentDebut(float montant, char* c_source, char* c_beneficiaire, char* date, int id) {
    virement* v = (virement*)malloc(sizeof(virement));
    if (v == NULL ) {
        printf("Erreur d'allocation mémoire.\n");
        return;
    }
    v->id = id;
    v->montant = montant;
    strcpy(v->compte_source, c_source);
    strcpy(v->compte_beneficiaire, c_beneficiaire);
    strcpy(v->date, date);

    v->prev = NULL;
    v->next = NULL;

    if (head == NULL) {
        head = v;
    }
     else {
        v->next = head ;
        head->prev = v ;
        head = v ;
    }

    printf(" Virement ajouté au début de la liste avec succès.\n");
}
```

/* Ajoute un virement à la fin de la liste

/*parcourt de la liste jusqu'au dernier élément.

```c
void ajouterViremmentFin(float montant, char* c_source, char* c_beneficiaire, char* date, int id) {
    virement* v = (virement*)malloc(sizeof(virement));
    if (v == NULL ) {
        printf("Erreur d'allocation mémoire.\n");
        return;
    }
    v->id = id;
    v->montant = montant;
    strcpy(v->compte_source, c_source);
    strcpy(v->compte_beneficiaire, c_beneficiaire);
    strcpy(v->date, date);

    v->prev = NULL;
    v->next = NULL;

    if (head == NULL) {
        head = v;
    } else {
        virement* tmp = head;
        while (tmp->next != NULL)
            tmp = tmp->next;
        v->prev = tmp;
        tmp->next = v;
    }

    printf(" Virement ajouté à la liste.\n");
}
```

```c
void ajouterApresID(int id_ref, float montant, char* c_source, char* c_beneficiaire, char* date, int id) {
    virement* temp = head;
    while (temp && temp->id != id_ref)
        temp = temp->next;

    if (!temp) {
        printf(" ID %d non trouvé.\n", id_ref);
        return;
    }

    virement* v = (virement*)malloc(sizeof(virement));
    if (v == NULL) {
        printf("Erreur d'allocation mémoire.\n");
        return;
    }

    v->id = id;
    v->montant = montant;
    strcpy(v->compte_source, c_source);
    strcpy(v->compte_beneficiaire, c_beneficiaire);
    strcpy(v->date, date);

    v->prev = temp;
    v->next = temp->next;

    if (temp->next != NULL)
        temp->next->prev = v;

    temp->next = v;

    printf(" Virement ajouté après l'ID %d.\n", id_ref);
}
```

/* On utilise cette fonction quand on veut insérer un virement juste après un autre qu'on identifie par son ID.

```c
void afficherVirements() {
    virement* tmp = head;
    if (tmp == NULL) {
        printf(" Aucun virement à afficher.\n");
        return;
    }
    printf(" Liste des virements :\n");
    while (tmp !=NULL) {
        printf("Virement : \n Id: %d \n compte source: %s \n compte beneficiaire: %s \n Date(YYYY-MM-DD): %s \n Montant: %.2f\n\n",
               tmp->id, tmp->compte_source, tmp->compte_beneficiaire, tmp->date, tmp->montant);
        tmp = tmp->next;
    }
}
```

/* Elle sert à afficher tous les virements enregistrés, avec tous leurs détails. Ça permet de tout visualiser facilement.

```c
void modifierVirement(int id  ,float montant,char* source,char* beneficiaire,char* date) {
    virement* tmp = head;

    while (tmp != NULL && tmp->id != id) {
        tmp = tmp->next;
    }

    if (tmp == NULL) {
        printf(" Aucun virement trouvé avec l'ID %d.\n", id);
        return;
    }

    printf(" Modification du virement avec ID %d :\n", id);

    printf("Nouveau montant : ");
    scanf("%f", &tmp->montant);

    printf("Nouveau compte source (sans espaces) : ");
    scanf("%s", tmp->compte_source);

    printf("Nouveau compte bénéficiaire (sans espaces) : ");
    scanf("%s", tmp->compte_beneficiaire);

    printf("Nouvelle date (YYYY-MM-DD) : ");
    scanf("%s", tmp->date);

    printf(" Virement modifié avec succès.\n");
}
```

/* cette fonction nous permets de corriger à partir de son ID Si on s'est trompé dans les infos d'un virement.

```c
//on va emplier pour la pile
void empiler(virement vire) {
    pile* nouvelElement = (pile*)malloc(sizeof(pile));
    if (!nouvelElement) {
        printf(" Erreur d'allocation mémoire.\n");
        return;
    }

    nouvelElement->data = vire;
    nouvelElement->next = sommet;
    sommet = nouvelElement;
}
```

/*Quand on supprime un virement, on le met de côté dans une pile grâce à cette fonction. Comme ça, on peut le récupérer plus tard si besoin.

```c
void supprimerDebut() {
    if (head == NULL) {
        printf(" Liste vide. Rien à supprimer.\n");
        return;
    }

    virement* tmp = head;
    head = head->next;

    if (head != NULL)
        head->prev = NULL;

        empiler(*tmp);
    printf("🗑 Virement avec ID %d supprimé du début.\n", tmp->id);
    free(tmp);
}
```

/* Elle supprime le tout premier virement de la liste, en le sauvegardant d'abord dans la pile.

```c
void supprimerFin() {
    if (head == NULL) {
        printf(" Liste vide. Rien à supprimer.\n");
        return;
    }

    virement* tmp = head;

    if (tmp->next == NULL) {
        printf(" Virement avec ID %d supprimé de la fin.\n", tmp->id);
        free(tmp);
        head = NULL;
        return;
    }

    while (tmp->next != NULL)
        tmp = tmp->next;

        empiler(*tmp);
    tmp->prev->next = NULL;
    printf(" Virement avec ID %d supprimé de la fin.\n", tmp->id);
    free(tmp);
}
```

/*Pareil que la précédente, mais ici on supprime le dernier virement de la liste.

```c
void supprimerApresID(int id_ref) {
    virement* tmp = head;

    while (tmp != NULL && tmp->id != id_ref)
        tmp = tmp->next;

    if (tmp == NULL || tmp->next == NULL) {
        printf(" Impossible de supprimer : ID %d inexistant ou aucun virement après lui.\n", id_ref);
        return;
    }

    virement* toDelete = tmp->next;
    tmp->next = toDelete->next;

    if (toDelete->next != NULL)
        toDelete->next->prev = tmp;
        empiler(*tmp);
    printf(" Virement avec ID %d supprimé après ID %d.\n", toDelete->id, id_ref);
    free(toDelete);
}

```

/* Cette fonction supprime le virement qui se trouve juste après un ID donné.

```c
void rechercherParMontant(float montant) {
    virement* tmp = head;
    int trouve = 0;

    printf("\n Recherche des virements avec montant = %.2f\n", montant);

    while (tmp != NULL) {
        if (tmp->montant == montant) {
            printf("\n Virement trouvé :\n");
            printf("ID : %d\n", tmp->id);
            printf("Compte source : %s\n", tmp->compte_source);
            printf("Compte bénéficiaire : %s\n", tmp->compte_beneficiaire);
            printf("Date : %s\n", tmp->date);
            printf("Montant : %.2f\n", tmp->montant);
            trouve = 1;
        }
        tmp = tmp->next;
    }

    if (!trouve) {
        printf(" Aucun virement trouvé avec ce montant.\n");
    }
}
```

/*On utilise cette fonction pour retrouver tous les virements qui ont un certain montant.

```c
void rechercherParCompteSource(char* source) {
    virement* tmp = head;
    int trouve = 0;

    while (tmp != NULL) {
        if (strcmp(tmp->compte_source, source) == 0) {
            printf(" Virement trouvé :\n");
            printf("ID: %d | Montant: %.2f | Source: %s | Bénéficiaire: %s | Date: %s\n\n",
                   tmp->id, tmp->montant, tmp->compte_source, tmp->compte_beneficiaire, tmp->date);
            trouve = 1;
        }
        tmp = tmp->next;
    }

    if (!trouve) {
        printf(" Aucun virement trouvé pour le compte source : %s\n", source);
    }
}

```

/* cette fonctions sert à chercher les virements qui ont été envoyés depuis un compte spécifique.

```c
void rechercherParBeneficiaire(char* beneficiaire) {
    virement* tmp = head;
    int trouve = 0;

    while (tmp != NULL) {
        if (strcmp(tmp->compte_beneficiaire, beneficiaire) == 0) {
            printf(" Virement trouvé :\n");
            printf("ID: %d | Montant: %.2f | Source: %s | Bénéficiaire: %s | Date: %s\n\n",
                   tmp->id, tmp->montant, tmp->compte_source, tmp->compte_beneficiaire, tmp->date);
            trouve = 1;
        }
        tmp = tmp->next;
    }

    if (!trouve) {
        printf(" Aucun virement trouvé pour le bénéficiaire : %s\n", beneficiaire);
    }
}

```

/* Ici, on cherche les virements qui ont été envoyés vers un certain compte bénéficiaire.

```c
void rechercheAvancee(float montantMin, char* source, char* date) {
    virement* tmp = head;
    int trouve = 0;

    while (tmp != NULL) {
        if (tmp->montant >= montantMin &&
            strcmp(tmp->compte_source, source) == 0 &&
            strcmp(tmp->date, date) == 0) {

            printf(" Virement correspondant trouvé :\n");
            printf("ID: %d | Montant: %.2f | Source: %s | Bénéficiaire: %s | Date: %s\n\n",
                   tmp->id, tmp->montant, tmp->compte_source, tmp->compte_beneficiaire, tmp->date);
            trouve = 1;
        }
        tmp = tmp->next;
    }

    if (!trouve) {
        printf(" Aucun virement ne correspond aux critères avancés.\n");
    }
}
```

/* C'est une recherche plus complète, où on peut filtrer par plusieurs critères en même temps (montant, date, etc.).

```c
void restaurerDernierViremementSupprime() {
    if (sommet == NULL) {
        printf(" Aucune suppression à annuler (pile vide).\n");
        return;
    }

    virement vire = sommet->data;

    pile* temp = sommet;
    sommet = sommet->next;
    free(temp);

    ajouterViremementFin(vire.montant, vire.compte_source, vire.compte_beneficiaire, vire.date, vire.id);

    printf(" Virement ID %d restauré avec succès.\n", vire.id);
}
```

/* Si on a supprimé un virement par erreur, cette fonction permet de le récupérer depuis la pile et de le remettre dans la liste.

```c
void afficherVirementsSupprimes() {
    if (sommet == NULL) {
        printf(" Aucun virement supprimé à afficher.\n");
        return;
    }

    pile* tmp = sommet;
    printf(" Virements supprimés (pile) :\n");

    while (tmp != NULL) {
        virement v = tmp->data;
        printf("ID: %d | Source: %s | Bénéficiaire: %s | Date: %s | Montant: %.2f\n",
               v.id, v.compte_source, v.compte_beneficiaire, v.date, v.montant);
        tmp = tmp->next;
    }
}

```

/* Elle affiche tous les virements qu'on a supprimés, donc ceux qui sont dans la pile.

```c
void effectuerVirement(int id) {
    virement* tmp = head;

    while (tmp != NULL && tmp->id != id) {
        tmp = tmp->next;
    }

    if (tmp == NULL) {
        printf(" Aucun virement trouvé avec l'ID %d.\n", id);
        return;
    }

    file* nouveau = (file*)malloc(sizeof(file));
    if (nouveau == NULL) {
        printf(" Erreur d'allocation mémoire pour la file.\n");
        return;
    }

    nouveau->data = *tmp;
    nouveau->next = NULL;

    if (debut == NULL) {
        debut = nouveau;
        fin = nouveau;
    } else {
        fin->next = nouveau;
        fin = nouveau;
    }

    if (tmp->prev != NULL)
        tmp->prev->next = tmp->next;
    else
        head = tmp->next;

    if (tmp->next != NULL)
        tmp->next->prev = tmp->prev;

    free(tmp);

    printf(" Virement exécuté et déplacé dans la file avec succès.\n");
}
```

/* Cette fonction simule l'exécution d'un virement en le déplaçant dans une file d'attente, comme si on l'avait réellement traité.

```c
void afficherFile() {
    file* tmp = debut;
    if (tmp == NULL) {
        printf(" Aucune opération en attente dans la file.\n");
        return;
    }

    printf(" Virements exécutés (dans la file) :\n");
    while (tmp != NULL) {
        printf("ID: %d | Montant: %.2f | Source: %s | Bénéficiaire: %s | Date: %s\n",
               tmp->data.id, tmp->data.montant, tmp->data.compte_source,
               tmp->data.compte_beneficiaire, tmp->data.date);
        tmp = tmp->next;
    }
}
```

/* Elle affiche les détails d'un seul virement. C'est pratique quand on veut voir les infos d'un virement spécifique sans afficher toute la liste.

```c
void demanderDonneesVirement(float *montant, char *source, char *beneficiaire, char *date, int *id) {
    printf("Entrer ID : ");
    scanf("%d", id);
    printf("Entrer montant : ");
    scanf("%f", montant);
    printf("Entrer compte source (sans espaces) : ");
    scanf("%s", source);
    printf("Entrer compte bénéficiaire (sans espaces) : ");
    scanf("%s", beneficiaire);
    printf("Entrer date (YYYY-MM-DD) : ");
    scanf("%s", date);
}
```

/* Cette fonction sert à demander à l'utilisateur de saisir toutes les informations d'un virement (montant, comptes, date, etc.). Elle facilite l'ajout de nouveaux virements de manière interactive.

Menu Principale

/* Le cœur de l'application repose sur la fonction main (), qui joue le rôle de menu interactif pour l'utilisateur. Elle permet d'accéder à toutes les fonctionnalités principales grâce à un système de switch qui oriente les choix selon les entrées de l'utilisateur. Le programme reste dans une boucle infinie pour que l'utilisateur puisse enchaîner plusieurs opérations sans redémarrer.

```
431
432  int main() {
433      int choixPrincipal, choixAjout, choixModifier, choixSupprimer, choixRechercher;
434      int id, idRef;
435      float montant;
436      char source[100], beneficiaire[100], date[100];
437
438      while (1) {
439          printf("\n===== MENU PRINCIPAL =====\n");
440          printf("1. Ajouter un virement\n");
441          printf("2. Modifier un virement\n");
442          printf("3. Supprimer un virement\n");
443          printf("4. Rechercher un virement\n");
444          printf("5. Effectuer un virement\n");
445          printf("6. Afficher tous les virements\n");
446          printf("7. Restaurer dernier virement supprimé\n");
447          printf("0. Quitter\n");
448          printf("Votre choix : ");
449          scanf("%d", &choixPrincipal);
450
451          switch (choixPrincipal) {
452              case 1:
```

```
450
451          switch (choixPrincipal) {
452              case 1:
453                  printf("\n-- Ajouter un virement --\n");
454                  printf("1. Au début\n");
455                  printf("2. Après un ID donné\n");
456                  printf("3. À la fin\n");
457                  printf("Votre choix : ");
458                  scanf("%d", &choixAjout);
459                  demanderDonneesVirement(&montant, source, beneficiaire, date, &id);
460                  switch (choixAjout) {
461                      case 1:
462                          ajouterViremementDebut(montant, source, beneficiaire, date, id);
463                          break;
464                      case 2:
465                          printf("Entrer l'ID après lequel ajouter : ");
466                          scanf("%d", &idRef);
467                          ajouterApresID(idRef, montant, source, beneficiaire, date, id);
468                          break;
469                      case 3:
470                          ajouterViremementFin(montant, source, beneficiaire, date, id);
471                          break;
472                      default:
473                          printf("Choix invalide.\n");
474                  }
475                  break;
476
```

```
476
477             case 2:
478                 printf("\n-- Modifier un virement --\n");
479                 printf("Entrer l'ID du virement à modifier : ");
480                 scanf("%d", &id);
481                 modifierVirement(id, 0, NULL, NULL, NULL);
482                 break;
483
```

```
483
484             case 3:
485                 printf("\n-- Supprimer un virement --\n");
486                 printf("1. Au début\n");
487                 printf("2. Après un ID donné\n");
488                 printf("3. À la fin\n");
489                 printf("Votre choix : ");
490                 scanf("%d", &choixSupprimer);
491                 switch (choixSupprimer) {
492                     case 1:
493                         supprimerDebut();
494                         break;
495                     case 2:
496                         printf("Entrer l'ID après lequel supprimer : ");
497                         scanf("%d", &idRef);
498                         supprimerApresID(idRef);
499                         break;
500                     case 3:
501                         supprimerFin();
502                         break;
503                     default:
504                         printf("Choix invalide.\n");
505                 }
506                 break;
507
```

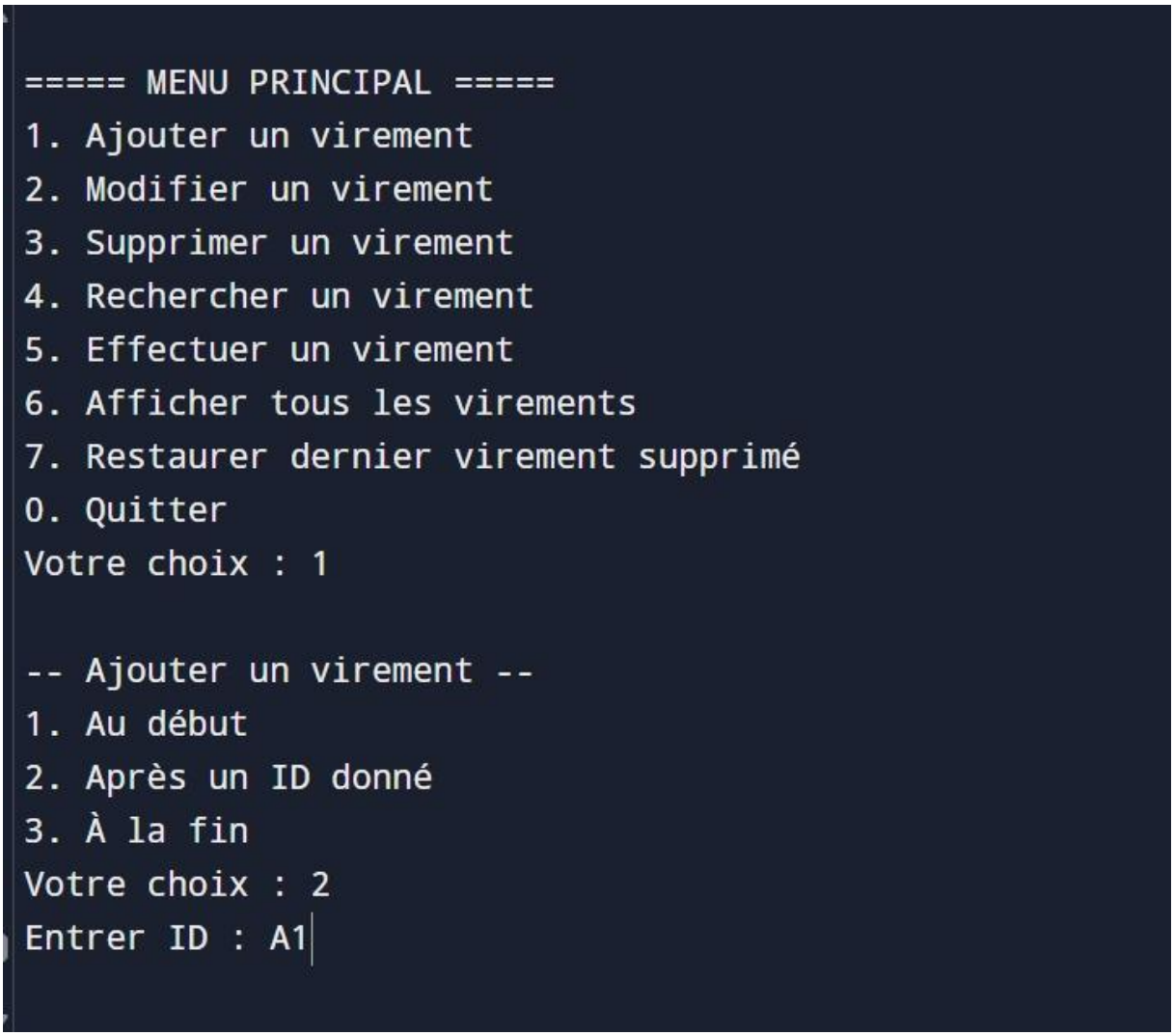
```
            case 4:
                printf("\n-- Rechercher un virement --\n");
                printf("1. Par montant\n");
                printf("2. Par compte source\n");
                printf("3. Par compte bénéficiaire\n");
                printf("4. Recherche avancée\n");
                printf("Votre choix : ");
                scanf("%d", &choixRechercher);
                switch (choixRechercher) {
                    case 1:
                        printf("Entrer le montant recherché : ");
                        scanf("%f", &montant);
                        rechercherParMontant(montant);
                        break;
                    case 2:
                        printf("Entrer le compte source : ");
                        scanf("%s", source);
                        rechercherParCompteSource(source);
                        break;
                    case 3:
                        printf("Entrer le compte bénéficiaire : ");
                        scanf("%s", beneficiaire);
                        rechercherParBeneficiaire(beneficiaire);
                        break;
                    case 4:
```

```
                        scanf("%s", source);
                        rechercherParCompteSource(source);
                        break;
                    case 3:
                        printf("Entrer le compte bénéficiaire : ");
                        scanf("%s", beneficiaire);
                        rechercherParBeneficiaire(beneficiaire);
                        break;
                    case 4:
                        printf("Montant minimum : ");
                        scanf("%f", &montant);
                        printf("Compte source : ");
                        scanf("%s", source);
                        printf("Date (YYYY-MM-DD) : ");
                        scanf("%s", date);
                        rechercheAvancee(montant, source, date);
                        break;
                    default:
                        printf("Choix invalide.\n");
                }
                break;
```

```
546             case 5:
547                 printf("\n-- Effectuer un virement --\n");
548                 printf("Entrer l'ID du virement à effectuer : ");
549                 scanf("%d", &id);
550                 effectuerVirement(id);
551                 break;
552 
553             case 6:
554                 afficherVirements();
555                 break;
556 
557             case 7:
558                 restaurerDernierViremementSupprime();
559                 break;
560 
561             case 0:
562                 printf("Au revoir !\n");
563                 exit(0);
564 
565             default:
566                 printf("Choix invalide.\n");
567         }
568     }
569 
570     return 0;
571 }
```

/* Fin du code.

Affichage des Résultats



```
Votre choix : 1

-- Ajouter un virement --
1. Au début
2. Après un ID donné
3. À la fin
Votre choix : 3

Entrer ID : 01

Entrer montant : 2000.00

Entrer compte source (sans espaces) : B34575

Entrer compte bénéficiaire (sans espaces) : A35678

Entrer date (YYYY-MM-DD) : 2025/25/2
 Virement ajouté à la liste.
```

NB : Le code sera exécuté en présence du professeur pour vérifier le bon fonctionnement de toutes les fonctionnalités développées.

Analyse

Organisation générale

Le programme repose sur un **menu principal** contenu dans la fonction main() qui regroupe l'ensemble des fonctionnalités proposées à l'utilisateur. Ce menu permet une navigation simple et efficace entre les différentes opérations disponibles : ajout, suppression, modification, recherche, exécution, affichage et restauration de virements.

Chaque fonctionnalité est associée à une **fonction spécifique**, ce qui nous a permis de découper le code en modules logiques. Cette organisation nous a beaucoup aidés pour répartir le travail entre les membres du groupe et faciliter les tests indépendants de chaque partie.

Logique fonctionnelle

Ajout de virements : Trois méthodes sont proposées **(début, après un ID donné, fin),** ce qui offre une bonne souplesse dans la construction de la liste.

Suppression et restauration : Les virements supprimés sont stockés dans une **pile**, ce qui permet une récupération rapide du dernier élément en cas d'erreur. Ce mécanisme a été utile pour gérer des suppressions accidentelles.

Modification : Elle se fait par ID. Nous avons volontairement restreint cette fonction pour éviter des modifications trop larges sans contrôle.

Recherche : Nous avons ajouté plusieurs types de recherches pour répondre à différents scénarios réels. L'option « recherche avancée » combine plusieurs critères, ce qui démontre la puissance de la structure.

Exécution des virements : Lorsqu'un virement est effectué, il est ajouté à une file, pour respecter l'ordre chronologique d'exécution, comme dans un système bancaire réel.

Affichage et sortie : Les virements peuvent être affichés à tout moment, et l'utilisateur peut quitter proprement l'application via l'option 0.

Bilan du travail de groupe

Ce projet nous a aidés à **mieux comprendre comment fonctionnent les listes doublement chaînées**, et comment on peut les utiliser dans un vrai programme.

On a aussi appris à utiliser des **structures comme les piles et les files** pour gérer les virements supprimés ou effectués, ce qui permet de **simuler un vrai système bancaire**.

Conclusion

Ce projet de gestion de virements bancaires nous a permis de mettre en pratique nos connaissances en langage C, tout en développant des compétences en structuration de données à travers l'utilisation de listes chaînées, de piles et de files.

Au-delà de la programmation, ce travail nous a également appris à collaborer efficacement en groupe, à répartir les tâches selon les compétences de chacune, et à chercher des solutions de manière autonome grâce aux ressources disponibles en ligne et aux outils informatiques.

Même si certaines étapes nous ont semblé complexes, la progression a été enrichissante et formatrice. Nous avons appris à surmonter les difficultés techniques, à valider notre code par des tests réguliers et à documenter notre travail de façon rigoureuse.

Ce mini-projet représente donc pour nous une étape importante dans notre parcours de formation, en nous préparant progressivement à des projets plus complexes à venir.

Bibliographie

Supports pédagogiques :

- ✓ Cours et travaux dirigés (TD) dispensés par nos enseignants en programmation C.
- ✓ Notes personnelles issues des explications données en classe.
- ✓ Échanges avec des étudiants ayant réalisé des projets similaires les années précédentes.
- ✓ Consultation de projets C disponibles en ligne à titre d'exemple.

Outils technologiques

- **Visual Studio Code** – Éditeur de code principal.
- **GCC Compiler** – Pour la compilation et l'exécution du code.