Projet Langage C

Membres du Groupe

Anna NDOYE

Nicoué Robert Valère DJAHLIN-NICOUE

Fatou Kiné THIOUB

Mame Diarra MBACKE

Salamata DIEDHIOU

Thierno Abdoulaye SALL

Lien vers le github :
https://github.com/mdmbest/ProjetC/tree/main

Table des matières

PARTIE 1 : Gestion des Étudiants (Chaînage par tableau)

1. Introduction

Dans le cadre de l'exercice 6 de la série 1 portant sur le chaînage par tableaux, nous avons développé une application permettant de gérer les résultats d'étudiants à un examen. Les étudiants sont représentés par une structure contenant leur numéro, leur nom et leur note. Plusieurs fonctionnalités ont été mises en œuvre : ajout, suppression, sauvegarde/restauration et affichage selon plusieurs critères (ordre de mérite, alphabétique, aléatoire).

2. Fonctionnalités implémentées

2.1 Ajouter un étudiant

```c
void ajouterUnEtudiant(Etudiant VETU[], int *n) {
    if (*n >= 99) {
        printf("Nombre maximal d'etudiants atteint.\n");
        return;
    }

    int num;
    printf("Numero de l'etudiant : ");
    scanf("%d", &num);

    // Vérification de l'unicité du numéro
    for (int i = 1; i <= *n; i++) {
        if (VETU[i].num == num) {
            printf("Ce numero est deja attribue a un autre etudiant.\n");
            return;
        }
    }

    VETU[*n + 1].num = num;
    getchar(); // Consommer le '\n' laissé par scanf

    printf("Nom complet de l'etudiant : ");
    fgets(VETU[*n + 1].nom, sizeof(VETU[*n + 1].nom), stdin);

    // Supprimer le retour à la ligne à la fin si présent
    size_t len = strlen(VETU[*n + 1].nom);
    if (len > 0 && VETU[*n + 1].nom[len - 1] == '\n') {
        VETU[*n + 1].nom[len - 1] = '\0';
    }

    printf("Note de l'etudiant : ");
    scanf("%f", &VETU[*n + 1].note);

    (*n)++;
    printf("Etudiant ajoute avec succes !\n");
}
```

La fonction ajouterUnEtudiant permet à l'utilisateur d'enregistrer un nouvel étudiant dans le tableau. Elle vérifie l'unicité du numéro d'étudiant et demande le nom ainsi que la note. Une confirmation est demandée pour ajouter plusieurs étudiants à la suite.

2.2 Supprimer un étudiant

```c
void supprimerUnEtudiant(Etudiant VETU[], int *n, int num) {
    int i, found = 0;

    // Parcours du tableau à partir de l'indice 1
    for (i = 1; i <= *n; i++) {
        if (VETU[i].num == num) {
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Étudiant non trouvé.\n");
        return;
    }

    // Décalage des éléments pour supprimer l'étudiant
    for (int j = i; j < *n; j++) {
        VETU[j] = VETU[j + 1];
    }

    (*n)--;

    printf("Etudiant supprime avec succes !\n");
}
```

La fonction supprimerUnEtudiant permet de rechercher un étudiant par son numéro puis de le retirer du tableau en décalant les éléments suivants. Une vérification est faite pour confirmer l'existence de l'étudiant avant suppression.

2.3 Sauvegarder les données

```c
void sauvegarderEtudiants(Etudiant VETU[], int n, char *fichier) {
    FILE *f = fopen(fichier, "w");
    if (!f) {
        printf("Erreur d'ouverture du fichier !\n");
        return;
    }

    for (int i = 1; i <= n; i++) {
        fprintf(f, "%d;%s;%.2f\n", VETU[i].num, VETU[i].nom, VETU[i].note);
    }

    fclose(f);
    printf("Sauvegarde réussie dans %s !\n", fichier);
}
```

La fonction sauvegarderEtudiants écrit les données de tous les étudiants dans un fichier texte. Chaque ligne du fichier contient le numéro, le nom et la note séparés par des points-virgules.

2.4 Restaurer les données

```c
void restaurerEtudiants(Etudiant VETU[], int *n, char *fichier) {
    FILE *f = fopen(fichier, "r");
    if (!f) {
        printf("Erreur d'ouverture du fichier !\n");
        return;
    }

    int i = 1; // Commence à 1
    while (fscanf(f, "%d;%[^;];%f\n", &VETU[i].num, VETU[i].nom, &VETU[i].note) == 3) {
        i++;
    }

    *n = i - 1;
    fclose(f);
    printf("Restauration réussie depuis %s !\n", fichier);
}
```

La fonction restaurerEtudiants lit les données depuis un fichier texte et les insère dans le tableau des étudiants au lancement du programme. Cela permet de reprendre l'état de la base après une précédente session.

2.5 Afficher les étudiants

- Trois types d'affichage sont proposés :
- - Par ordre alphabétique : les noms sont triés indépendamment de la casse grâce à la fonction strcasecmp.

```c
void afficherEtudiantsAlphabetique(Etudiant VETU[], int n) {
    // Tri alphabétique insensible à la casse
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcasecmp(VETU[i].nom, VETU[j].nom) > 0) {
                Etudiant temp = VETU[i];
                VETU[i] = VETU[j];
                VETU[j] = temp;
            }
        }
    }

    // Affichage des entêtes
    printf("\n\t\t\t+-----------+----------------------+------------+");
    printf("\n\t\t\t|   ID      |        NOM           |    NOTE    |");
    printf("\n\t\t\t+-----------+----------------------+------------+");

    // Affichage des étudiants triés
    for (int i = 0; i < n; i++) {
        char id_str[10], note_str[10];
        snprintf(id_str, sizeof(id_str), "%d", VETU[i].num);
        snprintf(note_str, sizeof(note_str), "%.2f", VETU[i].note);
        char nom_trunc[21] = {0};
        strncpy(nom_trunc, VETU[i].nom, 20);

        printf("\n\t\t\t| %-9s | %-20s | %-10s |", id_str, nom_trunc, note_str);
    }

    printf("\n\t\t\t+-----------+----------------------+------------+\n");
}
```

- - Par ordre de mérite : les étudiants sont chaînés via un tableau d’indices, ordonnés de la meilleure note à la moins bonne.

```c
void afficherEtudiantsMerite(Etudiant VETU[], int SUIVANT[], int DEB) {
    printf("\n\t\t\t+-----------+----------------------+-----------+");
    printf("\n\t\t\t|   ID      |        NOM           |    NOTE   |");
    printf("\n\t\t\t+-----------+----------------------+-----------+");

    int index = DEB;
    while (index != 0) {
        // Formatage des champs pour un alignement correct
        char id_str[10];
        char note_str[10];

        snprintf(id_str, sizeof(id_str), "%d", VETU[index].num);
        snprintf(note_str, sizeof(note_str), "%.2f", VETU[index].note);

        // Tronquer le nom si trop long
        char nom_trunc[21] = {0};
        strncpy(nom_trunc, VETU[index].nom, 20);

        printf("\n\t\t\t| %-9s | %-20s | %-10s |",
            id_str, nom_trunc, note_str);

        index = SUIVANT[index];
    }

    printf("\n\t\t\t+-----------+----------------------+-----------+\n");
}
```

- - De manière aléatoire : un mélange de Fisher-Yates est appliqué aux indices des étudiants pour un affichage non déterministe.

```c
void afficherEtudiantsAleatoire(Etudiant VETU[], int n) {
    int indices[n + 1]; // Assez d'espace pour les indices 1 à n
    for (int i = 1; i <= n; i++) {
        indices[i] = i;
    }

    // Mélange de Fisher-Yates en partant de l'indice 1
    srand(time(NULL));
    for (int i = n; i > 1; i--) {
        int j = 1 + rand() % (i); // entre 1 et i
        int temp = indices[i];
        indices[i] = indices[j];
        indices[j] = temp;
    }

    printf("\n\t\t\t+-----------+----------------------+------------+");
    printf("\n\t\t\t|   ID      |        NOM           |    NOTE    |");
    printf("\n\t\t\t+-----------+----------------------+------------+");

    for (int i = 1; i <= n; i++) {
        int idx = indices[i];
        char id_str[10], note_str[10];
        snprintf(id_str, sizeof(id_str), "%d", VETU[idx].num);
        snprintf(note_str, sizeof(note_str), "%.2f", VETU[idx].note);
        char nom_trunc[21] = {0};
        strncpy(nom_trunc, VETU[idx].nom, 20);

        printf("\n\t\t\t| %-9s | %-20s | %-10s |", id_str, nom_trunc, note_str);
    }

    printf("\n\t\t\t+-----------+----------------------+------------+\n");
}
```

3. Interface et navigation

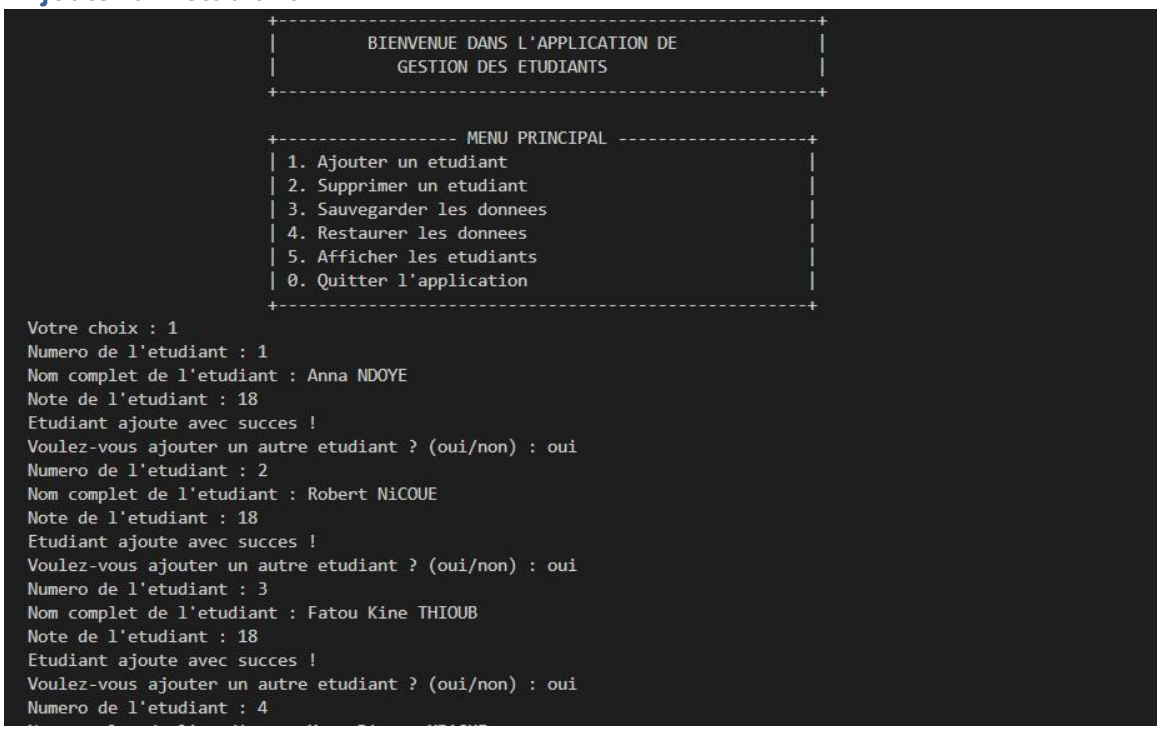
Le programme propose un menu principal interactif qui permet de naviguer entre les différentes options. Un sous-menu est dédié à l'affichage selon les critères précisés. L'utilisateur peut quitter l'application à tout moment et les données sont persistées automatiquement selon les choix de sauvegarde/restauration.

4. Compilation

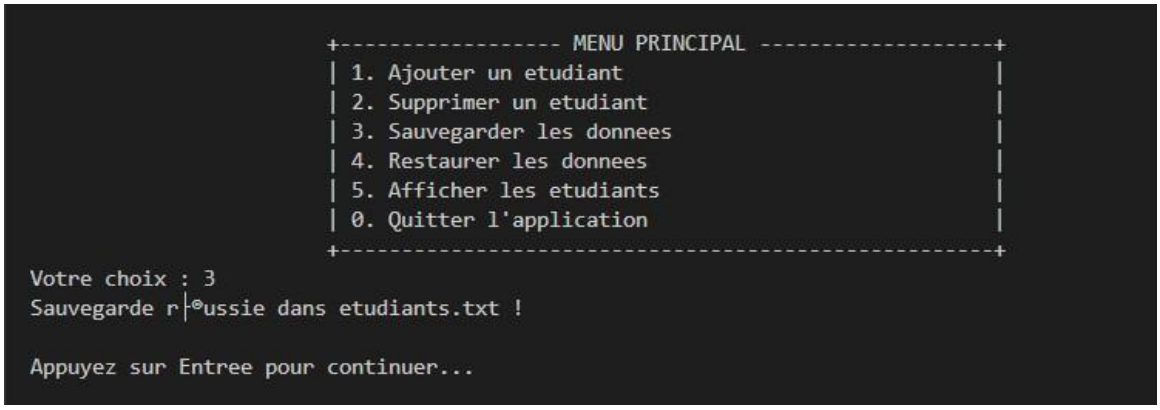
Un fichier batch nommé compiler.bat permet de compiler tous les fichiers nécessaires automatiquement. Il contient la ligne de compilation GCC suivante :
gcc main.c affichageEtudiantsAlphabetiques.c afficherEtudiantsAleatoires.c afficherParOrdreMerite.c ajouterEtudiant.c restaurerEtudiants.c sauvegarderEtudiants.c suprimerEtudiant.c tri_etudiants.c -o main.exe
Cela facilite la mise en production et l'exécution du programme avec ./main.exe.

Exécution du programme

1-Ajouter u n étudiant



2-sauvegarder les données



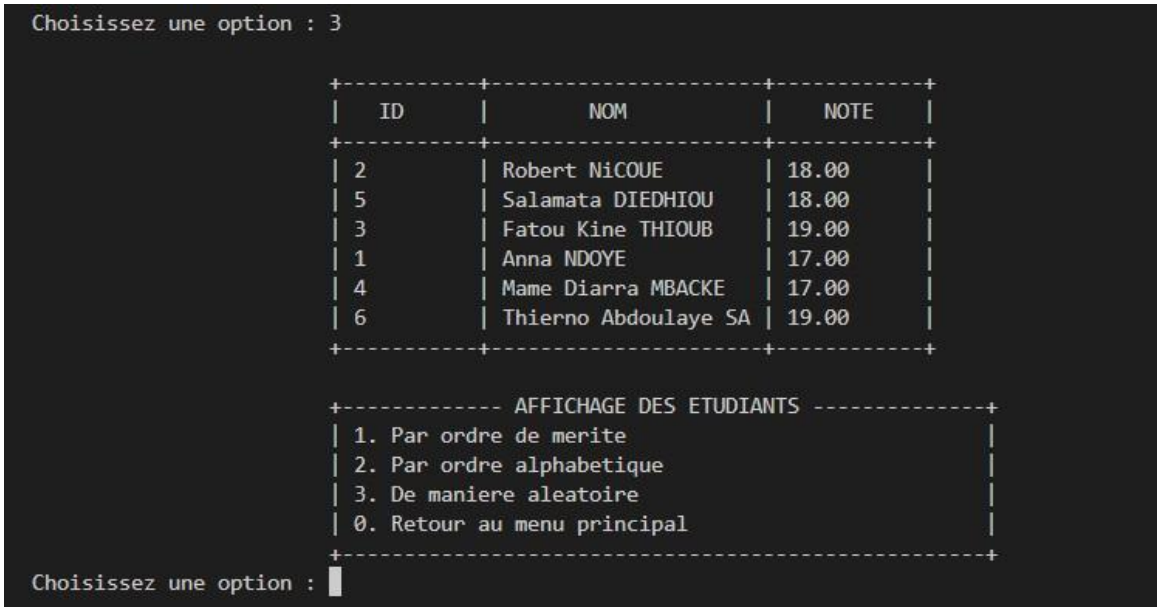
3-restaurer les données

```
Restauration r├®ussie depuis etudiants.txt !

Appuyez sur Entree pour continuer...

                        +------------------ MENU PRINCIPAL -------------------+
                        | 1. Ajouter un etudiant                              |
                        | 2. Supprimer un etudiant                            |
                        | 3. Sauvegarder les donnees                          |
                        | 4. Restaurer les donnees                            |
                        | 5. Afficher les etudiants                           |
                        | 0. Quitter l'application                            |
                        +-----------------------------------------------------+
Votre choix : 5
```

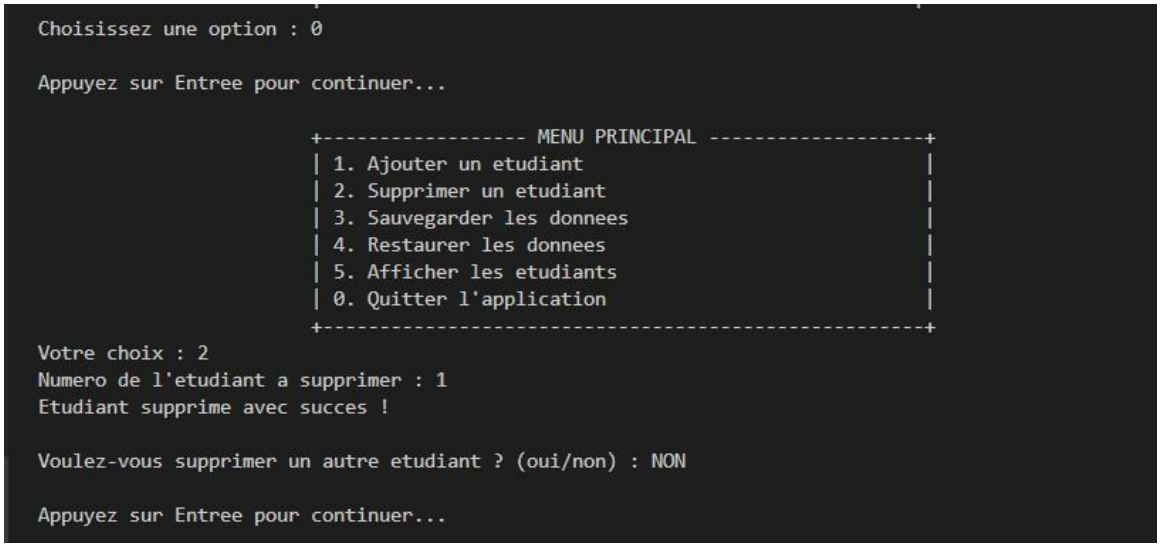
4-Afficher par ordre de mérite

```
                      +------------------ MENU PRINCIPAL -------------------+
                      | 1. Ajouter un etudiant                              |
                      | 2. Supprimer un etudiant                            |
                      | 3. Sauvegarder les donnees                          |
                      | 4. Restaurer les donnees                            |
                      | 5. Afficher les etudiants                           |
                      | 0. Quitter l'application                            |
                      +-----------------------------------------------------+
Votre choix : 5

                      +------------- AFFICHAGE DES ETUDIANTS --------------+
                      | 1. Par ordre de merite                              |
                      | 2. Par ordre alphabetique                           |
                      | 3. De maniere aleatoire                             |
                      | 0. Retour au menu principal                         |
                      +-----------------------------------------------------+
Choisissez une option : 1

                      +-----------+----------------------+------------+
                      |   ID      |        NOM           |    NOTE    |
                      +-----------+----------------------+------------+
                      | 3         | Fatou Kine THIOUB    | 19.00      |
                      | 6         | Thierno Abdoulaye SA | 19.00      |
                      | 5         | Salamata DIEDHIOU    | 18.00      |
                      | 2         | Robert NiCOUE        | 18.00      |
                      | 4         | Mame Diarra MBACKE   | 17.00      |
                      | 1         | Anna NDOYE           | 17.00      |
                      +-----------+----------------------+------------+
```

5-Afficher de manière aléatoire



6-Supprimer un étudiant



5. Conclusion

Cette première partie du projet démontre la capacité à structurer une application complète en C en utilisant des tableaux chaînés et une gestion simple des fichiers.

PARTIE 2 : Manipulation de SUPERMATRICES

1. Introduction

Dans cette deuxième partie du projet, nous avons implémenté et testé plusieurs opérations sur une structure appelée SUPERMATRICE, représentant une matrice dynamique en langage C. Ces fonctions permettent la gestion flexible des matrices : allocation, multiplication, permutation, extraction, conversion, vérification de la contiguïté mémoire et libération. L'objectif est d'appliquer des concepts fondamentaux de programmation mémoire et d'optimiser la manipulation de matrices.

2. Fonctions implémentées dans le fichier Supermat.c

2.1 aLLouerSupermat

```c
SUPERMRT aLLouerSupermat(int nL, int nc) {
    SUPERMRT sm = (SUPERMRT)malloc(sizeof(*sm));
    if (sm == NULL) return NULL; // L'allocation a échoué

    sm->nl = nL;
    sm->nc = nc;

    sm->ligne = (double **)malloc(nL * sizeof(double *));
    if (sm->ligne == NULL) {
        free(sm);
        return NULL; // L'allocation a échoué
    }

    for (int i = 0; i < nL; i++) {
        sm->ligne[i] = (double *)malloc(nc * sizeof(double));
        if (sm->ligne[i] == NULL) {
            for (int j = 0; j < i; j++) free(sm->ligne[j]);
            free(sm->ligne);
            free(sm);
            return NULL; // L'allocation a échoué
        }
    }

    return sm;
}
```

Cette fonction alloue dynamiquement une supermatrice de dimensions QL x Qc. Elle crée un tableau de pointeurs vers des lignes de double.
Exemple : SUPERMRT A = aLLouerSupermat(3, 3); Crée une matrice 3x3 vide.

2.2 superProduit

```c
// 3 MULTIPLICATION DE SUPERMATRICES
SUPERMRT superProduit(SUPERMRT a, SUPERMRT b) {
    if (a->nc != b->nl) return NULL;

    SUPERMRT c = aLLouerSupermat(a->nl, b->nc);
    if (!c) return NULL;

    for (int i = 0; i < a->nl; i++) {
        for (int j = 0; j < b->nc; j++) {
            acces(c, i, j) = 0.0;
            for (int k = 0; k < a->nc; k++) {
                acces(c, i, j) += acces(a, i, k) * acces(b, k, j);
                // c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return c;
}
```

Effectue le produit matriciel classique entre deux supermatrices a et b, à condition que le nombre de colonnes de a soit égal au nombre de lignes de b.
Exemple : A = [[1,2], [3,4]] ; B = [[5,6], [7,8]] ; Résultat : [[19, 22], [43, 50]]

2.3 permuterLignes

```c
void permuterLignes(SUPERMRT a, int i, int j)
{
    double *temp = a->ligne[i];
    a->ligne[i] = a->ligne[j];
    a->ligne[j] = temp;
}
```

Permute deux lignes i et j dans une supermatrice. Elle échange uniquement les pointeurs des lignes.
Exemple : Avant : Ligne 0 = [1,2,3], Ligne 1 = [4,5,6] ; Après permutation(0,1) : Ligne 0 = [4,5,6], Ligne 1 = [1,2,3]

2.4 sousMatrice

```c
SUPERMRT sousMatrice(SUPERMRT a, int L1, int L2, int C1, int C2) {
    // Vérification des bornes
    if (L1 < 0 || L2 >= a->nl || C1 < 0 || C2 >= a->nc || L1 > L2 || C1 > C2)
        return NULL;
        // Vérification de la validité des indices

    // Allocation du descripteur de la sous-matrice
    SUPERMRT sub = (SUPERMRT)malloc(sizeof(*sub));
    if (!sub) return NULL;

    // Définition des dimensions
    sub->nl = L2 - L1 + 1; //L2 =3 et L1 = 1 donc 3-1+1=3
    sub->nc = C2 - C1 + 1; //C2 = 4 et C1 = 2 donc 4-2+1=3

    // Allocation de la table des lignes
    sub->ligne = (double **)malloc(sub->nl * sizeof(double *));
    if (!sub->ligne) {
        free(sub);
        return NULL;
    }

    // Réutilisation des données existantes
    for (int i = 0; i < sub->nl; i++)
    {
        sub->ligne[i] = &a->ligne[L1 + i][C1];
    }

    return sub;
}
```

Extrait une sous-matrice à partir d'une supermatrice sans dupliquer les données. Elle utilise des pointeurs pour référencer les bonnes zones mémoire.

Exemple : A = [[1,2,3], [4,5,6], [7,8,9]] ; sousMatrice(A, 1, 2, 1, 2) → [[5,6], [8,9]]

2.5 matSupermat

```c
SUPERMRT matSupermat(double *m, int nLd, int ncd, int nLe, int nce) {
    SUPERMRT sm = (SUPERMRT)malloc(sizeof(*sm));
    if (sm == NULL) return NULL; // L'allocation a échoué

    sm->nl = nLe;
    sm->nc = nce;

    sm->ligne = (double **)malloc(nLe * sizeof(double *));
    if (sm->ligne == NULL) {
        free(sm);
        return NULL; // L'allocation a échoué
    }

    for (int i = 0; i < nLe; i++)
    {
        sm->ligne[i] = m + i * ncd; // Ligne i
    }

    return sm;
}
```

Convertit une matrice simple (tableau 1D) en une supermatrice en initialisant les pointeurs vers les bonnes adresses du tableau initial.

Exemple : double m[6] = {1,2,3,4,5,6}; matSupermat(m, ..., 2, 3) → [[1,2,3], [4,5,6]]

2.6 supermatMat

```c
void supermatMat(SUPERMRT sm, double *m, int nLd, int ncd)
{
    for (int i = 0; i < sm->nl; i++)
    {
        for (int j = 0; j < sm->nc; j++)
        {
            m[i * ncd + j] = acces(sm, i, j); //sm[i][j]
            // m[i][j] = sm->ligne[i][j];
        }


    }
}
```

Effectue l'opération inverse de matSupermat. Elle copie les valeurs d'une supermatrice dans un tableau 1D.

2.7 contiguite

```c
int contiguite(SUPERMRT a)
{
    int contigu = 1; // Supposons nue les lignes sont contiguës mais potentiellement désordonnées

    for (int i = 1; i < a->nl; i++)
    {
        if (a->ligne[i] == a->ligne[i - 1] + a->nc)
        {
            continue; // Toujours dans l'ordre attendu
        }
        else if (a->ligne[i] > a->ligne[0] && (a->ligne[i] - a->ligne[0]) % a->nc == 0)
        {
            contigu = 1; // Contigu mais hors de l'ordre attendu
        }
        else
        {
            return 0; // Pas contigu du tout
        }
    }
    return contigu == 1 ? 1 : 2;
}
```

Vérifie la contiguïté mémoire d'une supermatrice : Retourne 2 si elle est contiguë et ordonnée, 1 si contiguë mais désordonnée, 0 sinon.

2.8 rendreSupermat

```c
void reQdreSupermat(SUPERMRT sm)
{
    if (sm)
    {
        if (sm->ligne)
        {
            free(sm->ligne[0]);
            free(sm->ligne);
        }
        free(sm);
    }
}
```

Libère la mémoire allouée par une supermatrice. Si les lignes sont allouées de façon contiguë, elle libère seulement ligne[0] et ligne.

Dans le fichier Supermat.h on a les declarations des fonctions

```c
#ifndef SUPERMATRICES_H
#define SUPERMATRICES_H

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

typedef struct {
    int nl;       // Nombre de lignes
    int nc;       // Nombre de colonnes
    double **ligne; // Tableau de pointeurs vers les lignes
} *SUPERMRT;

// 2- Macros
#define acces(a, i, j) (a->ligne[i][j])

// Prototypes des fonctions
SUPERMRT aLlouerSupermat(int nL, int nc);
SUPERMRT superProduit(SUPERMRT a, SUPERMRT b);
void permuterLignes(SUPERMRT a, int i, int j);
SUPERMRT sousMatrice(SUPERMRT a, int L1, int L2, int c1, int c2);
SUPERMRT matSupermat(double *m, int nLd, int ncd, int nLe, int nce);
void supermatMat(SUPERMRT sm, double *m, int nLd, int ncd);
int contiguite(SUPERMRT a);
void rendreSupermat(SUPERMRT sm);

#endif
```

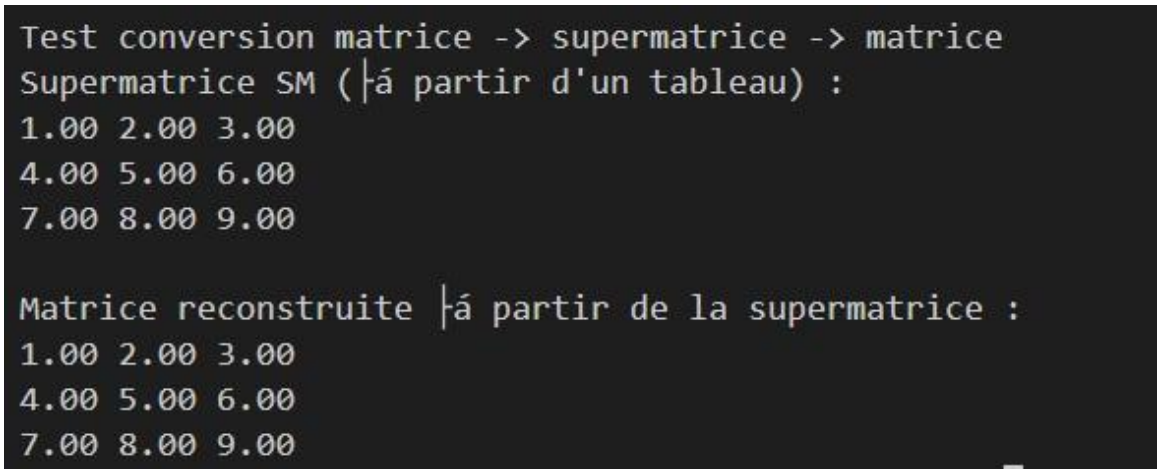
3. Résultats obtenus

Le fichier main.c contient les tests unitaires des fonctions :

- Affichage de A (matrice 3×3 de valeurs i+1 * j+1)
  1.00 2.00 3.00
2.00 4.00 6.00
  3.00 6.00 9.00
- Produit A * B avec B de même taille (3×3) : Produit matriciel correctement calculé. - Permutation des lignes 0 et 1 de A : Les lignes ont bien été échangées sans perte de données.
- Extraction de sous-matrice de A
- Vérification de la contiguïté : Selon le mode d'allocation, la fonction identifie si les lignes sont contiguës.

Exécution du programme

```
Matrice A :
1.00 2.00 3.00
2.00 4.00 6.00
3.00 6.00 9.00

Matrice B :
2.00 3.00 4.00
3.00 4.00 5.00
4.00 5.00 6.00

Produit A * B :
20.00 26.00 32.00
40.00 52.00 64.00
60.00 78.00 96.00

Permutation des lignes 0 et 1 dans A :
2.00 4.00 6.00
1.00 2.00 3.00
3.00 6.00 9.00

Sous-matrice de A (0,1) -> (1,2) :
2.00 4.00 6.00
1.00 2.00 3.00

V├®rification de la contigu├»t├® de A : 0

Test conversion matrice -> supermatrice -> matrice
Supermatrice SM (├á partir d'un tableau) :
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 9.00
```



4. Analyse

Ces fonctions démontrent une gestion avancée de la mémoire dynamique. L'utilisation de pointeurs sur lignes permet une grande flexibilité, notamment pour l'extraction de sous-matrices sans recopie, ce qui est très performant. Le respect de la contiguïté est fondamental pour optimiser les accès mémoire en C, et cette architecture permet de vérifier et contrôler cela.
En termes de conception, l'approche modulaire avec des macros (comme acces(a, i, j)) rend le code plus lisible et maintenable.

5. Conclusion

Cette deuxième partie montre la puissance des pointeurs en C dans la manipulation de structures de données dynamiques. Le concept de SUPERMATRICE est une abstraction pour manipuler de grandes matrices en toute sécurité tout en optimisant la mémoire.