

Partie 1

1- Ajouter un étudiant

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

- VETU[] : tableau d'étudiants
- *n : pointeur vers le nombre d'étudiants actuellement enregistrés (on le modifie ici)

On vérifie Si on a atteint 99 étudiants, si c'est le cas on arrête immédiatement.

```
int num;
printf("Numéro de l'étudiant : ");
scanf("%d", &num);

// Vérification de l'unicité du numéro
for (int i = 1; i <= *n; i++) {
    if (VETU[i].num == num) {
        printf("Ce numéro est déjà attribué à un autre étudiant.\n");
        return;
    }
}
```

- On demande à l'utilisateur un **identifiant numérique** pour l'étudiant.
- Parcours du tableau VETU du 1er étudiant jusqu'au n-ième pour voir si le numéro est déjà utilisé.
- Si trouvé, on **annule l'ajout**.

```
VETU[*n + 1].num = num;
```

L'étudiant est enregistré à la position suivante (*n + 1).

```
getchar(); // Consommer le '\n' laissé par scanf
```

On supprime le \n restant dans le buffer pour que fgets() fonctionne bien ensuite.

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

- Utilisation de fgets() pour lire **tout le nom avec des espaces**.
- Il permet de lire une chaîne de caractères contenant des espaces depuis l'entrée clavier (stdin) et de la stocker dans VETU[*n + 1].nom.
- **Détail des paramètres :**
 - VETU[*n + 1].nom : c'est la variable où on stocke le nom de l'étudiant.
 - sizeof(VETU[*n + 1].nom) : indique la taille maximale qu'on peut lire pour éviter de dépasser la mémoire allouée.
 - stdin : indique qu'on lit depuis l'entrée standard (le clavier).
- **Pourquoi utiliser fgets ?**
 - Parce que scanf("%s", ...) s'arrête au premier espace, ce qui ne permettrait pas d'écrire un nom comme "Fatou Ndiaye".
 - fgets() lit toute la ligne jusqu'à ce qu'elle soit trop longue ou qu'on appuie sur Entrée (\n).

```
size_t len = strlen(VETU[*n + 1].nom);
```

strlen(...) donne la **longueur** de la chaîne (combien de caractères y compris le \n si présent). On stocke la longueur dans **len**

```
if (len > 0 && VETU[*n + 1].nom[len - 1] == '\n') {
```

On vérifie s'il y a au moins un caractère (évite une erreur si la chaîne est vide), et si le **dernier caractère** est un saut de ligne \n.

```
VETU[*n + 1].nom[len - 1] = '\0';
```

Si oui, on le remplace par '\0', le caractère de fin de chaîne en C.

Résultat : la chaîne devient une chaîne de caractère, sans saut de ligne à la fin.

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

- On demande à l'utilisateur d'entrer la **note** de l'étudiant.

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

- On a ajouté un étudiant, donc on **incrémente** le nombre total d'étudiants.
- On Confirme à l'utilisateur que l'étudiant a été ajouté.

2- Supprimer un étudiant

```
void supprimerUnEtudiant(Etudiant VETU[], int *n, int num) {
```

VETU[] : tableau des étudiants.

*n : pointeur vers le nombre actuel d'étudiants.

num : numéro d'identification de l'étudiant à supprimer

```
int i, found = 0;
```

i : servira à parcourir le tableau.

found : indicateur pour savoir si l'étudiant a été trouvé (0 = non trouvé, 1 = trouvé).

```
for (i = 1; i <= *n; i++) {
    if (VETU[i].num == num) {
        found = 1;
        break;
}
```

Parcourt le tableau d'étudiants à partir de l'indice 1 jusqu'à *n.

Si l'étudiant avec ce num est trouvé, on sauvegarde sa position dans i et on met found = 1.

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

Si found vaut 0, on affiche un message d'erreur et on quitte la fonction.

```
for (int j = i; j < *n; j++) {  
    VETU[j] = VETU[j + 1];  
}  
  
(*n)--;
```

À partir de l'indice de l'étudiant à supprimer (i), on **décale tous les étudiants suivants vers la gauche**, pour **écraser** l'entrée à supprimer.

- On diminue n d'1 car un étudiant a été supprimé.

Exemple :

- Supposons n = 3 et on supprime l'étudiant à la position 2 :
 - VETU[2] = VETU[3]
 - VETU[3] est ensuite ignoré car n--

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

- Confirme que l'opération a été effectuée avec succès.

3- Sauvegarder les données

```
void sauvegarderEtudiants(Etudiant VETU[], int n, char *fichier) {
```

- VETU[] : tableau des étudiants.
- n : nombre d'étudiants actuellement dans le tableau.
- fichier : nom du fichier dans lequel sauvegarder les données (ex : "etudiants.txt").

```
FILE *f = fopen(fichier, "w");
```

Ouvre le fichier en mode **écriture ("w")** :

- Si le fichier existe déjà, il sera **écrasé**.
- Si le fichier n'existe pas, il sera **créé**.

f est un pointeur de type FILE qui permet de manipuler le fichier ouvert.

```
if (!f) {
    printf("Erreur d'ouverture du fichier !\n");
    return;
}
```

- Si l'ouverture du fichier a échoué (`f == NULL`), on affiche une erreur et on quitte la fonction.

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

Boucle à partir de `i = 1` jusqu'à `n`

Pour chaque étudiant, on écrit une ligne dans le fichier :

- Format : num;nom;note
- Exemple : 5;Mame Diarra;17.50

`%2f` : affiche la note avec **2 chiffres après la virgule**.

`fprintf()` fonctionne comme `printf()` mais écrit dans un fichier.

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

On ferme **le fichier** après utilisation pour garantir que toutes les données sont bien enregistrées et éviter les fuites de mémoire.

Affiche un message indiquant que tout s'est bien passé, avec le nom du fichier utilisé.

4- Restaurer les données

```
void restaurerEtudiants(Etudiant VETU[], int *n, char *fichier) {
```

`VETU[]` : tableau des étudiants à remplir.

`*n` : pointeur vers le nombre d'étudiants (on va le mettre à jour).

`fichier` : le fichier d'où on lit les données (ex : "etudiants.txt").

```
FILE *f = fopen(fichier, "r");
```

On tente d'ouvrir le fichier en **mode lecture** ("r").

```
if (!f) {  
    printf("Erreur d'ouverture du fichier !\n");  
    return;  
}
```

Si le fichier n'existe pas ou ne peut pas être lu, un message d'erreur s'affiche et on quitte la fonction.

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

On commence à $i = 1$ (encore une fois, indexation à partir de 1 dans ton code).

`fscanf(...)` == 3 signifie que 3 valeurs ont bien été lues :

- `%d` : le **numéro de l'étudiant** (ex : 10)
- `%[^;]` : le **nom** jusqu'au point-virgule (ex : Salamata Diédhio)
- `%f` : la **note** (ex : 17.50)

On lit une ligne comme : 10; Salamata Diédhio;17.50

On incrémente i pour passer au prochain étudiant.

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

Comme on a incrémenté i une fois de trop à la fin de la boucle, on fait $i - 1$ pour avoir le nombre correct d'étudiants chargés.

On ferme le fichier après lecture.

Confirmation à l'utilisateur que la restauration s'est bien passée.

5- Trier par ordre mérite

```
void trierEtudiantsMerite(Etudiant VETU[], int SUIVANT[], int n, int *DEB) {
```

VETU[] : tableau des étudiants.

SUIVANT[] : tableau de type liste chaînée (chaîne les indices).

n : nombre d'étudiants.

*DEB : pointeur vers l'indice du premier élément de la liste triée.

```
*DEB = 0; // Initialisation du debut à 0 (liste vide)
SUIVANT[0] = 0; // Dernier element pointe vers 0
```

On part avec une liste vide.

L'indice 0 est utilisé pour marquer la fin de la chaîne.

```
for (int i = 1; i <= n; i++) {
```

On parcourt tous les étudiants de 1 à n

```
if (*DEB == 0 || VETU[i].note > VETU[*DEB].note) {
    // Inserer au debut si la note est la plus elevee
    SUIVANT[i] = *DEB;
    *DEB = i;
```

Si la liste est vide ($*\text{DEB} == 0$) ou si $\text{VETU}[i]$ a une **meilleure note que le premier étudiant de la liste**, alors :

- On l'insère **au tout début de la liste**.
- Son SUIVANT pointe vers l'ancien début.
- Il devient le **nouveau début**.

```
else {
    // Trouver la bonne position en maintenant l'ordre decroissant
    int j = *DEB;
    while (SUIVANT[j] != 0 && VETU[SUIVANT[j]].note > VETU[i].note) {
        j = SUIVANT[j];
    }
    SUIVANT[i] = SUIVANT[j];
    SUIVANT[j] = i;
```

Sinon, on **cherche la bonne position** pour insérer l'étudiant i, de manière à ce que la liste reste triée par note décroissante :

- On avance dans la liste ($j = SUIVANT[j]$) tant que la note du suivant est **supérieure** à celle de i.
- Une fois la bonne position trouvée :
 - On insère i **entre j et $SUIVANT[j]$** .

Affichage par ordre de mérite

```
void afficherEtudiantsMerite(Etudiant VETU[], int SUIVANT[], int DEB) {
```

VETU[] : tableau des structures Etudiant.

SUIVANT[] : tableau qui simule une **liste chaînée**, avec les indices des "suivants".

DEB : indice du **début de la liste triée**.

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

On Affiche la **structure du tableau** avec des bordures et titres bien alignés :

- **ID** : Numéro d'étudiant.
- **NOM** : Nom de l'étudiant.
- **NOTE** : Sa note.

```
int index = DEB;  
while (index != 0) {
```

On commence par l'étudiant situé à l'indice DEB.

On continue jusqu'à ce que l'indice soit 0 (ce qui signifie **fin de liste**).

```

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);

```

On transforme les champs numériques num (entier) et note (float) en **chaînes de caractères** (char[]) :

- snprintf permet d'écrire les valeurs dans des buffers, en contrôlant leur taille.
- Cela permet un **alignement propre** dans le printf.

```

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

```

Pour éviter que le nom dépasse l'espace prévu (20 caractères), on copie au max les **20 premiers caractères**. On initialise à zéro pour éviter les caractères résiduels (\0 à la fin).

```

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

index = SUIVANT[index];

```

Affichage **avec alignement** :

- %9s : le numéro est affiché sur 9 caractères max.
- %20s : le nom sur 20.
- %10s : la note sur 10.

On **avance dans la liste triée**, en prenant l'indice suivant depuis SUIVANT[].

6- Trier par ordre alphabétique

```

void afficherEtudiantsAlphabetique(Etudiant VETU[], int n) {

```

Paramètres :

- VETU[] : tableau des étudiants.
- n : nombre d'étudiants à afficher.

```

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;
        }
    }
}

```

Triage par ordre alphabétique avec un **tri par sélection**.

strcasecmp() est utilisée pour comparer **sans tenir compte de la casse**.

- Par exemple, "Anna" et "anna" sont considérés comme identiques.

Si VETU[i].nom > VETU[j].nom, on échange les deux structures.

```

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

```

Affiche l'entête

```

for (int i = 0; i < n; i++) {

```

Parcourt tous les étudiants du tableau VETU.

i commence à 0 jusqu'à n - 1 (puisque le tableau commence à l'indice 0 ici).

```

char id_str[10], note_str[10];
snprintf(id_str, sizeof(id_str), "%d", VETU[i].num);

```

- Crée une **chaîne de caractères** (id_str) qui contient l'ID (num) de l'étudiant.
- Par exemple, si VETU[i].num = 123, alors id_str = "123".

On utilise snprintf Pourquoi ? Parce que printf() va afficher cette donnée dans un champ **formaté**, et on veut un alignement propre avec %-9s (9 caractères à gauche).

```
snprintf(note_str, sizeof(note_str), "%.2f", VETU[i].note);
```

- Convertit la **note de l'étudiant** en chaîne avec **2 chiffres après la virgule**.
- Par exemple, VETU[i].note = 15.75, devient note_str = "15.75".

Encore une fois, pour un **alignement propre** dans le tableau (10 caractères max).

```
char nom_trunc[21] = {0};
strncpy(nom_trunc, VETU[i].nom, 20);
```

- Crée une nouvelle chaîne nom_trunc qui contient **les 20 premiers caractères** du nom de l'étudiant.
- Pourquoi 20 ? Car la colonne du tableau est limitée à 20 caractères (%-20s).
- {0} initialise tout à \0 (fin de chaîne), pour éviter les erreurs si le nom est plus court.

Exemple : si le nom est "Anna Ndoye" → pas de souci.

Si le nom est plus long → il sera **coupé** à 20 caractères max.

7- Trier par ordre de aléatoire

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

On crée un tableau indices[] qui va contenir les positions de chaque étudiant

```
// Mélange des étudiants dans un ordre aléatoire
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;
}
```

On initialise le générateur aléatoire avec srand(time(NULL)) pour que le résultat change à chaque exécution.

Ensuite, on applique **l'algorithme de Fisher-Yates**, qui permet de mélanger les éléments sans doublons.

Cela change l'ordre des indices comme par exemple :

- avant : [1, 2, 3, 4, 5]
- après : [4, 1, 5, 3, 2]

```
for (int i = 1; i <= n; i++) {  
    int idx = indices[i];
```

On parcourt les **indices aléatoires** générés.

idx contient **l'indice réel** de l'étudiant dans VETU.

- Crée une **chaîne de caractères** (id_str) qui contient l'ID (num) de l'étudiant.
- Par exemple, si VETU[i].num = 123, alors id_str = "123".

L'ID (num) en chaîne id_str

La note en chaîne note_str avec 2 décimales

Le nom est **tronqué à 20 caractères max** pour un bon affichage

En utilisant snprintf

```
char id_str[10], note_str[10];  
snprintf(id_str, sizeof(id_str), "%d", VETU[i].num);
```

- Crée une **chaîne de caractères** (id_str) qui contient l'ID (num) de l'étudiant.
- Par exemple, si VETU[i].num = 123, alors id_str = "123".

On utilise snprintf Pourquoi ? Parce que printf() va afficher cette donnée dans un champ **formaté**, et on veut un alignement propre avec %-9s (9 caractères à gauche).

```
snprintf(note_str, sizeof(note_str), "%.2f", VETU[i].note);
```

- Convertit la **note de l'étudiant** en chaîne avec **2 chiffres après la virgule**.

- Par exemple, VETU[i].note = 15.75, devient note_str = "15.75".

Encore une fois, pour un **alignement propre** dans le tableau (10 caractères max).

```
char nom_trunc[21] = {0};
strncpy(nom_trunc, VETU[i].nom, 20);
```

- Crée une nouvelle chaîne nom_trunc qui contient **les 20 premiers caractères** du nom de l'étudiant.
- Pourquoi 20 ? Car la colonne du tableau est limitée à 20 caractères (%-20s).
- {0} initialise tout à \0 (fin de chaîne), pour éviter les erreurs si le nom est plus court.

Exemple : si le nom est "Anna Ndoye" → pas de souci.

Si le nom est plus long → il sera **coupé** à 20 caractères max.

8-Fichier main.c

```
#define MAX_ETUDIANTS 99
```

fixes ici le **nombre maximal d'étudiants** à 99.

```
void afficherMenuPrincipal() {
    printf("\n\t\t----- MENU PRINCIPAL -----+\n");
    printf("\t\t| 1. Ajouter un etudiant\n");
    printf("\t\t| 2. Supprimer un etudiant\n");
    printf("\t\t| 3. Sauvegarder les donnees\n");
    printf("\t\t| 4. Restaurer les donnees\n");
    printf("\t\t| 5. Afficher les etudiants\n");
    printf("\t\t| 0. Quitter l'application\n");
    printf("\t\t-----+\n");
}
```

- Menu principal contenant les **fonctionnalités principales** de l'appli (ajouter, supprimer, sauvegarder, etc.)

```

void afficherSousMenuAffichage() {
    printf("\n\t\t----- AFFICHAGE DES ETUDIANTS -----+\n");
    printf("\t\t| 1. Par ordre de mérite           |\n");
    printf("\t\t| 2. Par ordre alphabétique       |\n");
    printf("\t\t| 3. De manière aléatoire        |\n");
    printf("\t\t| 0. Retour au menu principal    |\n");
    printf("\t\t-----+\n");
}

```

- Options d'affichage : par mérite, par ordre alphabétique, ou aléatoire.

```

char demanderConfirmation(const char *message) {
    char reponse[10];

    // Vider le buffer résiduel pour éviter l'affichage de la question deux fois
    while (getchar() != '\n');

    do {
        printf("%s (oui/non) : ", message);
        fgets(reponse, sizeof(reponse), stdin);
        reponse[strcspn(reponse, "\n")] = '\0'; // Supprime le saut de ligne
        for (int i = 0; reponse[i]; i++) {
            reponse[i] = tolower(reponse[i]);
        }
    } while (strcmp(reponse, "oui") != 0 && strcmp(reponse, "non") != 0);

    return (strcmp(reponse, "oui") == 0) ? 'o' : 'n';
}

```

Affiche une question et attend une réponse "oui" ou "non".

Nettoie le buffer (getchar()) pour éviter les erreurs de lecture.

Convertit la réponse en minuscules avec tolower.

```

Etudiant ETU[NBETU]; // tableau d'étudiants
int SUIVANT[NBETU]; // tableau pour gérer les listes chaînées (tri par mérite)
int n = 0; // nombre d'étudiants actuellement
int DEB = 0; // début de la liste chaînée
char fichier[] = "etudiants.txt"; // fichier de sauvegarde

```

```

do {
    afficherMenuPrincipal();
    printf("Votre choix : ");

    if (scanf("%d", &choix) != 1) {
        while (getchar() != '\n'); // vider le buffer d'entrée
        printf("Entrée invalide. Veuillez entrer un nombre valide.\n");
        continue;
    }
    while (getchar() != '\n'); // vider correctement le buffer
}

```

b) Boucle principale avec do...while

- Tant que l'utilisateur ne choisit pas de quitter (choix != 0), on répète le menu.
- Utilise scanf() + getchar() pour gérer proprement les entrées.

```

case 1:
    if (n >= MAX_ETUDIANTS) {
        printf("Nombre maximal d'étudiants atteint !\n");
    } else {
        ajoutUnEtudiant(&ETU, &n);
        while (demanderConfirmation("Voulez-vous ajouter un autre étudiant ?") == 'o') {
            ajouterUnEtudiant(&ETU, &n);
        }
    }
break;

```

Appelle ta fonction pour ajouter un étudiant.

Tant que n < MAX_ETUDIANTS, demande si on veut ajouter un autre étudiant.

```

case 2:
    if (n == 0) {
        printf("Aucun étudiant à supprimer !\n");
    } else {
        do {
            int num;
            printf("Numéro de l'étudiant à supprimer : ");
            if (scanf("%d", &num) != 1) {
                while (getchar() != '\n');
                printf("Entrée invalide. Veuillez entrer un nombre valide.\n");
                continue;
            }
            while (getchar() != '\n');
            supprimerUnEtudiant(&ETU, &n, num);
        } while (n > 0 && demanderConfirmation("Voulez-vous supprimer un autre étudiant ?") == 'o');
    }
break;

```

Si des étudiants existent, demande leur numéro et les supprime.

Peut supprimer plusieurs étudiants à la suite si l'utilisateur le souhaite.

```
case 3:  
    sauvegarderEtudiants(VETU, n, fichier);  
    break;
```

- Sauvegarde les données actuelles dans un fichier texte.

```
case 4:  
    restaurerEtudiants(VETU, &n, fichier);  
    break;
```

- Récupère les données à partir du fichier de sauvegarde.

```
case 5:  
    do {  
        afficherSousMenuAffichage();  
        printf("Choisissez une option : ");  
        scanf(" %d", &sousChoix);  
        while (getchar() != '\n');  
  
        switch (sousChoix) {  
            case 1:  
                trierEtudiantsMerite(VETU, SUIVANT, n, &DEB);  
                afficherEtudiantsMerite(VETU, SUIVANT, DEB);  
                break;  
            case 2:  
                afficherEtudiantsAlphabetique(VETU, n);  
                break;  
            case 3:  
                afficherEtudiantsAleatoire(VETU, n);  
                break;  
            case 0:  
                break;  
            default:  
                printf("Option invalide, veuillez reessayer.\n");  
        }  
    } while (sousChoix != 0);  
    break;
```

Affiche un sous-menu :

- **1** : Affiche par ordre de mérite (avec liste chaînée via SUIVANT et DEB)
- **2** : Affiche par ordre alphabétique
- **3** : Affichage aléatoire
- **0** : Retour au menu principal

```
case 0:  
    printf("Merci d'avoir utilise l'application. A bientot !\n");  
    break;
```

- Affiche un message de sortie.

```
default:  
    printf("Option invalide, veuillez reessayer !\n");
```

- Gère les entrées invalides.

```
if (choix != 0) {  
    printf("\nAppuyez sur Entrée pour continuer...");  
    getchar();  
}
```

- Permet à l'utilisateur de lire le résultat avant de revenir au menu.