TD1 : Allocation Dynamique de Mémoire & Enregistrements

Exercice N° 1 :

1. Ecrire une fonction « **float moyenne(float *T, int taille)** » qui calcule et retourne la moyenne d'un tableau **T**.

```c
float moyenne(float *T, int taille) {
    float somme = 0;

    for (int i = 0; i < taille; i++) {
        somme += T[i];
    }

    return somme / taille;
}
```

2. Écrire un programme qui :

   a) Demande **n** le nombre d'étudiants,
   b) Alloue dynamiquement un tableau de **n** entiers,
   c) Saisit les `n` notes et affiche la moyenne,
   d) Demande `m` (nouveaux étudiants) et **agrandit** le tableau à `n+m`,
   e) Ré-affiche les notes et la nouvelle moyenne,
   f) Libère la mémoire avec **`free`**.

```c
int main() {
    int n, m;

    // a) Demander n
    printf("Donner le nombre d'etudiants : ");
    scanf("%d", &n);

    // b) Allocation dynamique
    float *notes = (float*) malloc(n * sizeof(float));
    if (notes == NULL) {
        printf("Erreur d'allocation !\n");
        return 1;
    }

    // c) Saisie et affichage de la moyenne
```

```c
    printf("Saisir les %d notes :\n", n);
    for (int i = 0; i < n; i++) {
        printf("Note %d : ", i+1);
        scanf("%f", &notes[i]);
    }

    printf("\nMoyenne = %.2f\n", moyenne(notes, n));

    // d) Demander m et agrandir le tableau
    printf("\nDonner le nombre de nouveaux etudiants m : ");
    scanf("%d", &m);

    notes = (float*) realloc(notes, (n + m) * sizeof(float));
    if (notes == NULL) {
        printf("Erreur de reallocation !\n");
        return 1;
    }

    // Saisie des nouvelles notes
    printf("Saisir les %d nouvelles notes :\n", m);
    for (int i = n; i < n + m; i++) {
        printf("Note %d : ", i+1);
        scanf("%f", &notes[i]);
    }

    // e) Ré-afficher les notes et la nouvelle moyenne
    printf("\nToutes les notes : ");
    for (int i = 0; i < n + m; i++) {
        printf("%.2f ", notes[i]);
    }

    printf("\nNouvelle moyenne = %.2f\n", moyenne(notes, n + m));

    // f) Libérer la mémoire
    free(notes);

    return 0;
}
```

Exercice N° 2 :

On souhaite gérer dynamiquement un tableau d'entiers **trié en ordre croissant**.

1. Écrire un programme qui :
    - demande à l'utilisateur le **nombre initial d'éléments** `n`,
    - alloue dynamiquement un tableau d'entiers de taille `n`,
    - lit les `n` entiers **déjà triés** par l'utilisateur,
    - affiche le tableau.
2. Écrire une fonction qui permet d'insérer un nouvel élément, la fonction doit :

- augmente la taille du tableau de 1 (avec `realloc`),
- insère la nouvelle valeur `val` **à la bonne position** pour que le tableau reste trié,
- met à jour `n`.

3. Dans le programme principal :
   - demander à l'utilisateur une valeur à insérer,
   - appeler la fonction `insererElement`,
   - afficher le tableau mis à jour (toujours trié).

Exemple d'exécution attendue :

```
Entrez le nombre d'éléments : 5
Entrez les 5 valeurs triées : 2 8 15 23 40
Tableau initial : 2 8 15 23 40
Entrez la valeur à insérer : 18
Tableau après insertion : 2 8 15 18 23 40
```

Exercice N° 3 :

Nous voulons développer une petite application pour gestion des étudiants.

1. Déclarer une structure **Etudiant** pour modéliser un étudiant (nom, age, note).
2. Créer une fonction « **Etudiant* lireEtudiant()** » qui permet de lire et retourner un étudiant.
3. Ecrire une fonction « **void afficherEtudiant(Etudiant etu)** » qui affiche les informations d'un étudiant passé en paramètre.
4. Ecrire une fonction « **float moyenne(Etudiant *et, int n)** » qui calcule et retourne la moyenne des **n** étudiants passés en paramètres.
5. Dans la fonction main(), demander à l'utilisateur le nombre n d'étudiants, allouer dynamiquement un tableau de n étudiants, lire leurs informations, puis afficher la liste et la moyenne générale.

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

// Déclaration de la structure Etudiant
typedef struct {
    char nom[30];
    int age;
    float note;
} Etudiant;

// Fonction qui lit et retourne un pointeur sur un étudiant
Etudiant* lireEtudiant() {
    Etudiant* e = (Etudiant*)malloc(sizeof(Etudiant));  // allocation
dynamique

    if (e == NULL) { // Vérification d'allocation
```

```
        printf("Erreur d'allocation mémoire !\n");
        exit(1);
    }

    printf("Entrez le nom de l'étudiant : ");
    scanf("%s", e->nom);

    printf("Entrez l'âge de l'étudiant : ");
    scanf("%d", &e->age);

    printf("Entrez la note de l'étudiant : ");
    scanf("%f", &e->note);

    return e;  // renvoie un pointeur sur la structure allouée
}

// Fonction d'affichage d'un étudiant
void afficherEtudiant(Etudiant etu) {
    printf("Nom : %s\t Âge : %d\t Note : %.2f\n", etu.nom, etu.age, etu.note);
}

// Fonction qui calcule la moyenne d'un tableau d'étudiants
float moyenne(Etudiant *et, int n) {
    float somme = 0;
    for (int i = 0; i < n; i++) {
        somme += et[i].note;
    }
    return somme / n;
}

// Fonction principale
int main() {
    int n;

    printf("Entrez le nombre d'étudiants : ");
    scanf("%d", &n);

    // Allocation dynamique d'un tableau de n étudiants
    Etudiant *tab = (Etudiant*)malloc(n * sizeof(Etudiant));

    if (tab == NULL) {
        printf("Erreur d'allocation mémoire !\n");
        return 1;
    }

    // Lecture des informations des étudiants
    for (int i = 0; i < n; i++) {
        printf("\n--- Étudiant %d ---\n", i + 1);
```

```
        Etudiant *temp = lireEtudiant(); // retourne un pointeur sur un
étudiant
        tab[i] = *temp;                  // copie le contenu dans le tableau
        free(temp);                      // libère la mémoire temporaire
    }

    // Affichage de la liste des étudiants
    printf("\n=== Liste des étudiants ===\n");
    for (int i = 0; i < n; i++) {
        afficherEtudiant(tab[i]);
    }

    // Calcul et affichage de la moyenne
    float moy = moyenne(tab, n);
    printf("\nMoyenne générale : %.2f\n", moy);

    // Libération de la mémoire
    free(tab);

    return 0;
}
```

Exercice N° 4 :

Écrire un programme en C qui :

1. Définit deux structures : une structure **Adresse** contenant : ville, rue, codePostal et une structure **Employe** contenant : nom et un champ adr de type Adresse.
2. Ecrire une fonction « **Employe* lireEmploye** » qui lit les informations d'un employé (nom et son adresse complète).
3. Affiche toutes les informations de l'employé à l'aide d'une fonction « **void afficherEmploye (Employe e)** ».
4. Une fonction **main** pour lire n employés et les afficher sur écran.

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

// Définition des structures

typedef struct {
    char ville[30];
    char rue[30];
    int codePostal;
} Adresse;

typedef struct {
```

```c
    char nom[30];
    Adresse adr;  // champ de type Adresse
} Employe;

// Fonction pour lire un employé
Employe* lireEmploye() {
    Employe *e = (Employe*)malloc(sizeof(Employe));

    if (e == NULL) {
        printf("Erreur d'allocation mémoire !\n");
        exit(1);
    }

    printf("Entrez le nom de l'employé : ");
    scanf("%s", e->nom);

    printf("Entrez la ville : ");
    scanf("%s", e->adr.ville);

    printf("Entrez la rue : ");
    scanf("%s", e->adr.rue);

    printf("Entrez le code postal : ");
    scanf("%d", &e->adr.codePostal);

    return e;
}

// Fonction pour afficher un employé
void afficherEmploye(Employe e) {
    printf("Nom : %s\n", e.nom);
    printf("Ville : %s\n", e.adr.ville);
    printf("Rue : %s\n", e.adr.rue);
    printf("Code postal : %d\n", e.adr.codePostal);
    printf("-----------------------------\n");
}

// Fonction principale
int main() {
    int n;
    printf("Entrez le nombre d'employés : ");
    scanf("%d", &n);

    Employe *tab = (Employe*)malloc(n * sizeof(Employe));
    if (tab == NULL) {
        printf("Erreur d'allocation mémoire !\n");
        return 1;
    }
```

```c
    // Lecture des employés
    for (int i = 0; i < n; i++) {
        printf("\n--- Employé %d ---\n", i + 1);
        Employe *temp = lireEmploye();
        tab[i] = *temp;  // copie du contenu
        free(temp);
    }

    // Affichage des employés
    printf("\n=== Liste des employés ===\n");
    for (int i = 0; i < n; i++) {
        afficherEmploye(tab[i]);
    }

    free(tab);
    return 0;
}
```