Allocation dynamique de la mémoire et pointeurs en C

1. Objectifs

- Comprendre le concept de mémoire dynamique
- Apprendre les fonctions d'allocation dynamique de mémoire en C
- Gérer correctement la mémoire dynamique
- Créer des structures de données dynamiques
- Appliquer l'allocation dynamique de mémoire à des problèmes concrets

2. Introduction à l'Allocation Dynamique de Mémoire

- L'allocation dynamique de mémoire permet de demander de la mémoire au moment de l'exécution, plutôt qu'au moment de la compilation. Cela est particulièrement utile lorsque la taille de la mémoire nécessaire n'est pas connue à l'avance.
- **Fonctions d'Allocation Dynamique**
- En C, la bibliothèque standard fournit plusieurs fonctions pour l'allocation dynamique de mémoire, disponibles dans `<stdlib.h>`.

3. Allocation Dynamique de Mémoire et Pointeurs en C

- En **C**, il est crucial d'initialiser un pointeur avant d'utiliser l'opérateur d'**indirection** *. Voici les deux principales façons d'initialiser un pointeur :
- *En lui affectant l'adresse d'une variable existante :*

```
int a = 10;  // Déclaration et initialisation d'une variable entière
int *p;// Déclaration d'un pointeur vers un entier
p = &a;// Initialisation du pointeur avec l'adresse de la variable 'a'
*p = 2;// Utilisation du pointeur pour modifier la valeur de 'a'
```

- *En réservant dynamiquement un espace mémoire :*
  - Pour pouvoir affecter directement une valeur à *p, il est nécessaire de réserver un espace mémoire adéquat pour stocker la valeur. Cela se fait à l'aide de fonctions d'allocation dynamique de mémoire telles que malloc ou calloc.
- *Exemple avec malloc :*

apcpedagogie.com

```
int *p = (int*)malloc(sizeof(int));  // Allocation dynamique de mémoire pour
un entier
if (p == NULL) {
    // Gestion de l'erreur si l'allocation échoue
    printf("Échec de l'allocation mémoire.\n");
    return 1;
}
*p = 2;  // Affectation de la valeur 2 à l'espace mémoire pointé par 'p'
```

4. Allocation Dynamique de Mémoire avec `malloc` en C

- La fonction `malloc` (memory allocation) est utilisée en **C** pour allouer de la mémoire dynamique.
- Elle est définie dans la bibliothèque <stdlib.h>. Voici une explication détaillée de la syntaxe et des exemples d'utilisation :
- ***Syntaxe de malloc***

```
nom_pointeur = (type_pointeur) malloc(nombre_octets);
nom_pointeur : Le nom du pointeur qui recevra l'adresse du bloc de mémoire
alloué.
type_pointeur : Le type de données du pointeur. La conversion de type (cast)
est nécessaire pour correspondre au type du pointeur.
nombre_octets : Le nombre d'octets à allouer. La fonction sizeof(type_objet)
est souvent utilisée pour déterminer ce nombre.
```

- ***Exemple 1 : Allocation d'un seul entier***

```
#include <stdlib.h>

int main() {
    int *p;// Déclaration d'un pointeur vers un entier
    p = (int *)malloc(sizeof(int)); // Allocation dynamique de mémoire pour un
entier
    if (p == NULL) {
        // Vérification de la réussite de l'allocation
        return 1;
    }
    *p = 10;// Affectation de la valeur 10 à l'espace mémoire alloué
    free(p);// Libération de la mémoire allouée
    return 0;
```

apcpedagogie.com

```
}
```

- ***Exemple 2 : Allocation et Utilisation d'un Entier***

```
#include
#include
#include

int main() {
    int i = 3;                      // Déclaration et initialisation d'un
entier
    int *p = NULL;                  // Initialisation d'un pointeur à NULL

    // Affichage de la valeur de p avant l'initialisation
    printf("Valeur de p avant initialisation = %p\n", (void *)p);

    p = (int *)malloc(sizeof(int)); // Allocation dynamique de mémoire pour un
entier

    // Affichage de la valeur de p après l'initialisation
    printf("Valeur de p après initialisation = %p\n", (void *)p);

    if (p == NULL) {
        // Vérification de la réussite de l'allocation
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    *p = i;                         // Affectation de la valeur de i à
l'espace mémoire alloué
    printf("Valeur de *p = %d\n", *p); // Affichage de la valeur de l'entier
pointé par p

    free(p);                        // Libération de la mémoire allouée
    getch();                        // Attente d'une entrée de l'utilisateur
    return 0;
}
```

- ***Explications des Résultats***
- Valeur de **p** avant initialisation :
  - Avant l'appel à malloc, p est initialisé à NULL, ce qui est une bonne pratique pour éviter les pointeurs indéfinis. L'affichage de NULL montre que p ne pointe vers aucune adresse valide avant l'allocation.

apcpedagogie.com

- Valeur de **p** après initialisation :
  - Après l'appel à malloc, p contient l'adresse du bloc de mémoire alloué. Cette adresse est une valeur non nulle (par exemple 0x11424), ce qui signifie que l'allocation a réussi.
- *Valeur de p :
  - Après l'affectation *p = i, la valeur pointée par p est 3, comme attendu, puisque i est initialisé à 3.
- Vérification de l'Allocation
- Il est toujours important de vérifier que malloc a réussi à allouer la mémoire demandée. Si malloc échoue, il renvoie NULL, ce qui peut être vérifié comme suit :

```
if (p == NULL) {
    printf("Échec de l'allocation mémoire.\n");
    return 1;
}
```

Libération de la Mémoire Toute mémoire allouée dynamiquement avec malloc doit être libérée une fois qu'elle n'est plus nécessaire, en utilisant la fonction free :

```
free(p);
```

- Libérer la mémoire évite les fuites de mémoire, qui peuvent conduire à une consommation excessive de mémoire et éventuellement à l'épuisement de la mémoire disponible.

5. Allocation Dynamique d'un Tableau en C

- L'allocation dynamique d'un tableau permet de créer des tableaux dont la taille est déterminée à l'exécution, ce qui est utile lorsque la taille n'est pas connue à l'avance.

1. **Exemple : Allocation Dynamique d'un Tableau**
   - Voici un exemple détaillé montrant comment allouer dynamiquement un tableau d'entiers en C.

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

int main() {
    int *p = NULL;  // Déclaration et initialisation du pointeur à NULL
```

```
    int i, n;

    printf("\nDonner le nombre d’éléments du tableau: ");
    scanf("%d", &n);  // Lecture du nombre d'éléments

    // Allocation dynamique de mémoire pour n entiers
    p = (int *)malloc(n * sizeof(int));
    if (p == NULL) {
        // Vérification de la réussite de l'allocation
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Initialisation des éléments du tableau à 0
    for (i = 0; i < n; i++) {
        *(p + i) = 0;  // Équivalent à p[i] = 0
    }

    // Affichage des éléments du tableau
    printf("Éléments du tableau après initialisation: ");
    for (i = 0; i < n; i++) {
        printf("%d ", p[i]);
    }
    printf("\n");

    free(p);  // Libération de la mémoire allouée

    return 0;
}
```

2. **Explications des Concepts**

1. **Déclaration et Initialisation :**

   - `int *p = NULL;`: Le pointeur p est initialisé à NULL. Cela permet d'éviter des comportements indéfinis en cas de tentative d'utilisation avant allocation.

2. **Lecture de la Taille du Tableau :**

   - `scanf("%d", &n);` : La fonction scanf lit la taille du tableau (le nombre d'éléments) entrée par l'utilisateur et la stocke dans n.

apcpedagogie.com

3. **Allocation Dynamique de Mémoire :**
    - `p = (int *)malloc(n * sizeof(int));`: La fonction malloc alloue n blocs de mémoire, chaque bloc ayant la taille d'un entier (sizeof(int)). La conversion de type (int *) est nécessaire pour que malloc retourne un pointeur de type int *.
4. **Vérification de l'Allocation :**
    - `if (p == NULL) { ... }` : Il est important de vérifier que malloc n'a pas renvoyé NULL, ce qui indiquerait un échec de l'allocation de mémoire.
5. **Initialisation des Éléments du Tableau :**
    - `for (i = 0; i < n; i++) { *(p + i) = 0; }`: Cette boucle initialise chaque élément du tableau à 0. L'expression `*(p + i)` est équivalente à `p[i]`.
6. **Affichage des Éléments du Tableau :**
    - `for (i = 0; i < n; i++) { printf("%d ", p[i]); }` : Cette boucle affiche les éléments du tableau après l'initialisation.
7. **Libération de la Mémoire :**
    - `free(p);`: La mémoire allouée dynamiquement doit être libérée avec free pour éviter les fuites de mémoire.

3. **Résumé**

- L'exemple montre comment allouer dynamiquement un tableau d'entiers dont la taille est spécifiée par l'utilisateur à l'exécution.
- Chaque élément du tableau est ensuite initialisé à 0. Voici les étapes principales :
    - Lecture de la taille du tableau.
    - Allocation dynamique de mémoire pour le tableau.
    - Initialisation des éléments du tableau.
    - Affichage des éléments du tableau.
    - Libération de la mémoire allouée.

apcpedagogie.com

- L'utilisation de l'allocation dynamique de mémoire permet de créer des structures de données flexibles qui peuvent s'adapter aux besoins de l'application à l'exécution.

6. Allocation Dynamique d'une Structure en C

- L'allocation dynamique de structures en C permet de créer des instances de structures dont la durée de vie et la taille peuvent être gérées dynamiquement.

1. **Exemples**

- Voici deux exemples pour illustrer comment allouer dynamiquement des structures et des tableaux de structures.

1. **Exemple :01**

- **Allocation Dynamique d'une Seule Structure**
- Ce premier exemple montre comment allouer de la mémoire pour une seule structure etudiant et comment saisir les champs de cette structure via un pointeur.

```
#include
#include

struct etudiant {
    int numins;
    char nom[20];
};

int main() {
    struct etudiant *p = NULL;

    // Allocation dynamique de mémoire pour une structure
etudiant
    p = (struct etudiant *)malloc(sizeof(struct etudiant));
    if (p == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Saisie des champs de la structure
    printf("\nDonner le numéro d'inscription de l'étudiant:
```

apcpedagogie.com

```
");
    scanf("%d", &(p->numins));

    printf("\nDonner le nom de l'étudiant: ");
    fflush(stdin);  // Vide le tampon d'entrée
    gets(p->nom);

    // Affichage des données saisies
    printf("\nNuméro d'inscription: %d", p->numins);
    printf("\nNom de l'étudiant: %s\n", p->nom);

    // Libération de la mémoire allouée
    free(p);

    return 0;
}
```

- **Exemple :02**
  - **Allocation Dynamique d'un Tableau de Structures**
  - Ce second exemple montre comment allouer de la mémoire pour un tableau de structures etudiant et comment saisir les champs de chaque structure dans le tableau via un pointeur.

```
#include
#include

struct etudiant {
    int numins;
    char nom[20];
};

int main() {
    struct etudiant *classe = NULL;
    int i, n;

    printf("\nDonner le nombre des étudiants: ");
    scanf("%d", &n);

    // Allocation dynamique de mémoire pour un tableau
de n structures etudiant
    classe = (struct etudiant *)malloc(n *
```

apcpedagogie.com

```
sizeof(struct etudiant));
    if (classe == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Saisie des champs de chaque structure dans le
tableau
    for (i = 0; i < n; i++) {
        printf("\nDonner le numéro d'inscription de
l'étudiant: ");
        scanf("%d", &((classe + i)->numins));

        printf("\nDonner le nom de l'étudiant: ");
        fflush(stdin);  // Vide le tampon d'entrée
        gets((classe + i)->nom);
    }

    // Affichage des données d'un étudiant spécifique
    printf("\nDonner l'indice de l'étudiant: ");
    scanf("%d", &i);
    if (i >= 0 && i < n) {
        printf("\nLe numéro d'inscription de l'étudiant
est %d", (classe + i)->numins);
        printf("\nLe nom de l'étudiant est %s\n",
(classe + i)->nom);
    } else {
        printf("\nIndice invalide\n");
    }

    // Libération de la mémoire allouée
    free(classe);

    return 0;
}
```

2. **Explications des Concepts**

- **Allocation Dynamique d'une Structure :**
  - La fonction malloc est utilisée pour allouer de la mémoire pour une structure. Le pointeur retourné est de type void*, il est donc nécessaire de le convertir au type approprié (par exemple, (struct etudiant *)).
  - Saisie et Accès aux Champs d'une Structure :

apcpedagogie.com

        - On utilise l'opérateur -> pour accéder aux champs d'une structure pointée par un pointeur.
    - **Allocation Dynamique d'un Tableau de Structures :**
        - La fonction `malloc` est utilisée pour allouer de la mémoire pour un tableau de structures. La taille totale allouée est n * sizeof(struct etudiant), où n est le nombre de structures dans le tableau.
        - Les éléments du tableau sont accédés en utilisant l'arithmétique des pointeurs, par exemple (classe + i)->numins.
    - **Libération de la Mémoire :**
        - Toute mémoire allouée dynamiquement doit être libérée avec free pour éviter les fuites de mémoire.

3. **Notes Importantes**

    - **Vérification de l'Allocation** : Toujours vérifier que malloc n'a pas renvoyé NULL, ce qui indiquerait un échec de l'allocation de mémoire.
    - **Utilisation de fflush(stdin)** : fflush(stdin) est souvent utilisé pour vider le tampon d'entrée avant de lire une chaîne de caractères avec gets. Notez que fflush(stdin) est un comportement non standard et n'est pas portable ; une alternative plus portable serait d'utiliser fgets à la place de gets.
    - Ces exemples montrent comment gérer la mémoire dynamique en C pour des structures, permettant une gestion flexible et efficace des données en fonction des besoins de l'application.

7. Fonction de Libération de la Mémoire : `free`

- En C, l'allocation dynamique de mémoire est une fonctionnalité puissante, mais elle nécessite une gestion explicite de la mémoire. Chaque fois que de la mémoire est allouée dynamiquement avec malloc, calloc, ou realloc, il est essentiel de libérer cette mémoire lorsque vous n'en avez plus besoin. Ceci permet d'éviter les fuites de mémoire, qui peuvent épuiser la mémoire disponible et conduire à des comportements inattendus du programme.

1. **Syntaxe de `free`**

```
free(nom_pointeur);
```

- `nom_pointeur` : Le pointeur qui fait référence à la mémoire allouée dynamiquement.

2. **Exemple: Utilisation de `malloc` et `free` avec une Structure**

   - Voici un exemple complet montrant comment allouer et libérer dynamiquement de la mémoire pour une seule structure ainsi qu'un tableau de structures.
   - *Exemple 1 : Allocation et Libération d'une Seule Structure*

```
#include
#include

struct etudiant {
    int numins;
    char nom[20];
};

int main() {
    struct etudiant *p = NULL;

    // Allocation dynamique de mémoire pour une structure etudiant
    p = (struct etudiant *)malloc(sizeof(struct etudiant));
    if (p == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Saisie des champs de la structure
    printf("\nDonner le numéro d'inscription de l'étudiant: ");
    scanf("%d", &(p->numins));

    printf("\nDonner le nom de l'étudiant: ");
    fflush(stdin);  // Vide le tampon d'entrée
    gets(p->nom);

    // Affichage des données saisies
    printf("\nNuméro d'inscription: %d", p->numins);
    printf("\nNom de l'étudiant: %s\n", p->nom);

    // Libération de la mémoire allouée
    free(p);

    return 0;
}
```

   - *Exemple 2 : Allocation et Libération d'un Tableau de Structures*

apcpedagogie.com

```
#include
#include

struct etudiant {
    int numins;
    char nom[20];
};

int main() {
    struct etudiant *classe = NULL;
    int i, n;

    printf("\nDonner le nombre des étudiants: ");
    scanf("%d", &n);

    // Allocation dynamique de mémoire pour un tableau de n structures
etudiant
    classe = (struct etudiant *)malloc(n * sizeof(struct etudiant));
    if (classe == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Saisie des champs de chaque structure dans le tableau
    for (i = 0; i < n; i++) {
        printf("\nDonner le numéro d'inscription de l'étudiant: ");
        scanf("%d", &((classe + i)->numins));

        printf("\nDonner le nom de l'étudiant: ");
        fflush(stdin);  // Vide le tampon d'entrée
        gets((classe + i)->nom);
    }

    // Affichage des données d'un étudiant spécifique
    printf("\nDonner l'indice de l'étudiant: ");
    scanf("%d", &i);
    if (i >= 0 && i < n) {
        printf("\nLe numéro d'inscription de l'étudiant est %d", (classe
+ i)->numins);
        printf("\nLe nom de l'étudiant est %s\n", (classe + i)->nom);
    } else {
        printf("\nIndice invalide\n");
```

apcpedagogie.com

```
    }

    // Libération de la mémoire allouée
    free(classe);

    return 0;
}
```

3. **Bonnes Pratiques pour l'Allocation Dynamique de Mémoire**

- ***Vérification de l'Allocation :***
  - Toujours vérifier que la mémoire allouée par malloc n'est pas NULL avant d'utiliser le pointeur. Cela évite les accès mémoire invalides.
- ***Libération de la Mémoire :***
  - Toujours appeler free pour libérer la mémoire allouée dynamiquement lorsque vous n'en avez plus besoin. Cela aide à éviter les fuites de mémoire.
- ***Initialisation des Pointeurs :***
  - Initialisez toujours les pointeurs à NULL après la libération de la mémoire pour éviter les accès à des emplacements de mémoire déjà libérés.
- ***Utilisation de fflush(stdin) :***
  - Utilisez fflush(stdin) avec prudence pour vider le tampon d'entrée. Notez que ce comportement n'est pas standard en C ; fflush est généralement utilisé pour les flux de sortie.
- ***Gestion des Erreurs :***
  - Gérez les erreurs d'allocation de mémoire de manière appropriée, par exemple, en affichant un message d'erreur et en terminant le programme ou en essayant de réallouer la mémoire.

4. **Conclusion**

- L'utilisation correcte de malloc pour l'allocation de mémoire et free pour la libération de mémoire est essentielle pour écrire des programmes C robustes et efficaces. La gestion dynamique de la mémoire permet une utilisation flexible des ressources, mais elle nécessite une attention particulière pour éviter les erreurs courantes telles que les fuites de mémoire et les accès invalides à la mémoire.

apcpedagogie.com

8. Applications

1. **App:01**

- Travail demandé
  - Ecrire un programme qui demande la taille d'un tableau d'entiers à créer, saisit les éléments du tableau au clavier, les affiche et affiche leur somme.
  - Le tableau sera créé de façon dynamique.

Solution

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

int main() {
    int *tableau = NULL;  // Pointeur pour le tableau d'entiers
    int taille, i, somme = 0;

    // Demande à l'utilisateur de saisir la taille du tableau
    printf("Donner la taille du tableau d'entiers: ");
    scanf("%d", &taille);

    // Allocation dynamique de mémoire pour le tableau
    tableau = (int *)malloc(taille * sizeof(int));
    if (tableau == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Saisie des éléments du tableau
    printf("Saisir les éléments du tableau:\n");
    for (i = 0; i < taille; i++) {
        printf("Élément %d: ", i + 1);
        scanf("%d", &tableau[i]);
    }

    // Affichage des éléments du tableau et calcul de la somme
    printf("Les éléments du tableau sont: ");
    for (i = 0; i < taille; i++) {
        printf("%d ", tableau[i]);
        somme += tableau[i];
    }
```

apcpedagogie.com

```
    printf("\n");

    // Affichage de la somme des éléments
    printf("La somme des éléments du tableau est: %d\n", somme);

    // Libération de la mémoire allouée
    free(tableau);

    return 0;
}
```

2. **App:02**

- Travail demandé
  - Soit la structure etudiant suivante :

```
typedef struct etudiant
{
char nom [20] ;
char prenom [10];
char mat [10] ;
float * Tmoy ;
float moyg ;
} ;
```

Avec :

- **nom** : le nom de l'étudiant
- **prenom**: le prénom de l'étudiant
- **mat** : le matricule de l'étudiant
- **Tmoy**: représente un pointeur vers un tableau qui va contenir les notes des matières d'un étudiant qui sera alloué dynamiquement.
- **moyg** : un champ qui va contenir la moyenne générale de l'étudiant

- **1.** Ecrire une fonction calcul_moyg qui retourne la moyenne générale d'un tableau donné en paramètre. (Les matières ont le même coefficient) `float calcul_moyg (float *Tmoy, int n)`
- **2.** Ecrire une fonction qui permet de saisir un étudiant : `struct etudiant saisie_etudiant ()`

apcpedagogie.com

- Cette fonction saisit le nom, prénom, matricule d'un étudiant. Elle demande le nombre de matières, alloue dynamiquement Tmoy et le remplit. Enfin, elle calcule la moyg à l'aide de calcule_moyg.
- **3.** Ecrire un programme principal qui :
  - demande le nombre des étudiants d'une classe et alloue dynamiquement un tableau d'étudiants.
  - Calcule la moyenne de la classe.
- Affiche matricule, nom, prénom de l'étudiant qui a la moyenne la plus élevée.

Solution

- Gestion des Étudiants et Calcul des Moyennes
- Nous allons écrire un programme en C qui gère une liste d'étudiants, calcule leur moyenne générale et trouve l'étudiant ayant la meilleure moyenne.
- Voici comment nous allons structurer le programme :
  - Définir la structure etudiant.
  - Écrire la fonction calcul_moyg qui calcule la moyenne générale des notes.
  - Écrire la fonction saisie_etudiant qui saisit les détails d'un étudiant et remplit les notes dynamiquement.
  - Écrire le programme principal qui demande le nombre d'étudiants, alloue dynamiquement un tableau d'étudiants, calcule la moyenne de la classe et affiche l'étudiant avec la meilleure moyenne.
- ***Définition de la Structure etudiant***

```
typedef struct etudiant {
    char nom[20];
    char prenom[10];
    char mat[10];
    float *Tmoy;
    float moyg;
} etudiant;
```

- ***Fonction calcul_moyg***
  - Cette fonction calcule la moyenne générale des notes contenues dans un tableau dynamique.

apcpedagogie.com

```
float calcul_moyg(float *Tmoy, int n) {
    float somme = 0.0;
    for (int i = 0; i < n; i++) {
        somme += Tmoy[i];
    }
    return (n > 0) ? (somme / n) : 0.0;
}
```

- ***Fonction saisie_etudiant***
- Cette fonction saisit les informations d'un étudiant, alloue dynamiquement la mémoire pour les notes, les remplit et calcule la moyenne générale.

```
#include
#include
#include

etudiant saisie_etudiant() {
    etudiant etu;
    int nb_matieres;

    // Saisie des informations de l'étudiant
    printf("Nom de l'étudiant: ");
    scanf("%s", etu.nom);

    printf("Prénom de l'étudiant: ");
    scanf("%s", etu.prenom);

    printf("Matricule de l'étudiant: ");
    scanf("%s", etu.mat);

    // Saisie du nombre de matières
    printf("Nombre de matières: ");
    scanf("%d", &nb_matieres);

    // Allocation dynamique de la mémoire pour les notes
    etu.Tmoy = (float *)malloc(nb_matieres * sizeof(float));
    if (etu.Tmoy == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        exit(1);
    }
```

apcpedagogie.com

```
    // Saisie des notes
    printf("Saisir les notes:\n");
    for (int i = 0; i < nb_matieres; i++) {
        printf("Note %d: ", i + 1);
        scanf("%f", &etu.Tmoy[i]);
    }

    // Calcul de la moyenne générale
    etu.moyg = calcul_moyg(etu.Tmoy, nb_matieres);

    return etu;
}
```

- ***Programme Principal***
- Ce programme demande le nombre d'étudiants, alloue dynamiquement un tableau d'étudiants, calcule la moyenne de la classe et affiche l'étudiant ayant la meilleure moyenne.

```
int main() {
    int nb_etudiants;
    etudiant *classe;
    float somme_moyennes = 0.0, moyenne_classe;
    int indice_meilleure_moyenne = 0;

    // Demande du nombre d'étudiants
    printf("Nombre d'étudiants: ");
    scanf("%d", &nb_etudiants);

    // Allocation dynamique de mémoire pour le tableau
d'étudiants
    classe = (etudiant *)malloc(nb_etudiants *
sizeof(etudiant));
    if (classe == NULL) {
        printf("Échec de l'allocation mémoire.\n");
        return 1;
    }

    // Saisie des informations de chaque étudiant et calcul
de la moyenne
    for (int i = 0; i < nb_etudiants; i++) {
        printf("\nSaisie des informations pour l'étudiant
%d:\n", i + 1);
```

apcpedagogie.com

```
        classe[i] = saisie_etudiant();
        somme_moyennes += classe[i].moyg;
        if (classe[i].moyg >
classe[indice_meilleure_moyenne].moyg) {
            indice_meilleure_moyenne = i;
        }
    }

    // Calcul de la moyenne de la classe
    moyenne_classe = somme_moyennes / nb_etudiants;

    // Affichage des résultats
    printf("\nMoyenne de la classe: %.2f\n", moyenne_classe);
    printf("Étudiant avec la meilleure moyenne:\n");
    printf("Matricule: %s\n",
classe[indice_meilleure_moyenne].mat);
    printf("Nom: %s\n",
classe[indice_meilleure_moyenne].nom);
    printf("Prénom: %s\n",
classe[indice_meilleure_moyenne].prenom);
    printf("Moyenne Générale: %.2f\n",
classe[indice_meilleure_moyenne].moyg);

    // Libération de la mémoire allouée pour les notes de
chaque étudiant et pour le tableau d'étudiants
    for (int i = 0; i < nb_etudiants; i++) {
        free(classe[i].Tmoy);
    }
    free(classe);

    return 0;
}
```

- ***Explications***
  - Déclaration et Allocation de la Mémoire :
    - Le programme demande le nombre d'étudiants et alloue dynamiquement un tableau de structures etudiant.
  - Saisie et Calcul :
    - Pour chaque étudiant, le programme utilise saisie_etudiant pour remplir les informations et calculer la moyenne générale.
    - La moyenne de la classe est calculée en additionnant les moyennes de tous les étudiants et en divisant par le nombre total d'étudiants.

apcpedagogie.com

- Affichage de l'Étudiant avec la Meilleure Moyenne :
  - Le programme parcourt le tableau d'étudiants pour trouver l'étudiant avec la meilleure moyenne et affiche ses informations.
- Libération de la Mémoire :
  - La mémoire allouée dynamiquement pour les notes de chaque étudiant et pour le tableau d'étudiants est libérée à la fin du programme pour éviter les fuites de mémoire.
  - Ce programme montre comment gérer la mémoire dynamique en C, manipuler des structures complexes et effectuer des calculs sur des données saisies par l'utilisateur.

Cours en C

Source : https://apcpedagogie.com/allocation-dynamique-de-la-memoire-et-pointeurs-en-c · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com