Exercices Avancés sur les Pointeurs en C

Exercice 9 : Gestion des pointeurs de fonctions

Énoncé :

Écrire un programme en C qui :

1. Déclare deux fonctions, `addition` et `multiplication`, qui prennent deux entiers en paramètres et renvoient leur somme et leur produit respectivement.

2. Déclare un pointeur de fonction.

3. Utilise le pointeur de fonction pour appeler dynamiquement `addition` et `multiplication` en fonction de l'entrée de l'utilisateur.

Solution :

```
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int multiplication(int a, int b) {
    return a * b;
}

int main() {
    int (*operation)(int, int); // Déclaration du pointeur de fonction
    int choix, x, y;

    printf("Choisissez l'opération : 1 pour addition, 2 pour multiplication : ");
```

```
    scanf("%d", &choix);
    printf("Entrez deux entiers : ");
    scanf("%d %d", &x, &y);

    if (choix == 1) {
        operation = addition;
    } else if (choix == 2) {
        operation = multiplication;
    } else {
        printf("Choix invalide.\n");
        return 1;
    }

    int resultat = operation(x, y);
    printf("Le résultat est : %d\n", resultat);

    return 0;
}
```

Exercice 10 : Manipulation de listes chaînées avec des pointeurs

Énoncé :

Écrire un programme en C qui :

1. Déclare une structure pour représenter un n?ud dans une liste chaînée (contenant un entier et un pointeur vers le n?ud suivant).

2. Implémente des fonctions pour ajouter un n?ud en tête, ajouter un n?ud en queue, et afficher tous les n?uds de la liste.

3. Utilise ces fonctions pour créer une liste chaînée et afficher son contenu.

Solution :

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

typedef struct Node {
    int data;
    struct Node *next;
} Node;

void ajouterEnTete(Node **head, int newData) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = newData;
    newNode->next = *head;
    *head = newNode;
}

void ajouterEnQueue(Node **head, int newData) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = newData;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node *temp = *head;
```

```
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void afficherListe(Node *node) {
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    Node *head = NULL;

    ajouterEnTete(&head, 10);
    ajouterEnTete(&head, 20);
    ajouterEnQueue(&head, 30);
    ajouterEnQueue(&head, 40);

    printf("Contenu de la liste : ");
    afficherListe(head);

    return 0;
```

```
}
```

Exercice 11 : Pointeurs et tableaux multidimensionnels

Énoncé :

Écrire un programme en C qui :

1. Déclare et initialise une matrice 3x3.

2. Utilise des pointeurs pour transposer la matrice.

3. Affiche la matrice originale et la matrice transposée.

Solution :

```
#include <stdio.h>


void afficherMatrice(int mat[3][3], int taille) {
    for (int i = 0; i < taille; i++) {
        for (int j = 0; j < taille; j++) {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }
}


void transpose(int (*mat)[3], int taille) {
    for (int i = 0; i < taille; i++) {
        for (int j = i; j < taille; j++) {
            int temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
```

```
        }
    }
}

int main() {
    int mat[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    printf("Matrice originale :\n");
    afficherMatrice(mat, 3);

    transpose(mat, 3);

    printf("Matrice transposée :\n");
    afficherMatrice(mat, 3);

    return 0;
}
```

Exercice 12 : Gestion de la mémoire avec `calloc` et `realloc`

Énoncé :

Écrire un programme en C qui :

1. Utilise `calloc` pour allouer dynamiquement un tableau d'entiers initialisé à zéro.

2. Utilise `realloc` pour redimensionner le tableau.

3. Remplit le tableau avec des valeurs données par l'utilisateur et affiche les éléments du tableau après chaque redimensionnement.

4. Libère la mémoire allouée dynamiquement.

Solution :

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

void afficherTableau(int *arr, int taille) {
    for (int i = 0; i < taille; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int tailleInitiale, nouvelleTaille;

    printf("Entrez la taille initiale du tableau : ");
    scanf("%d", &tailleInitiale);

    int *arr = (int *)calloc(tailleInitiale, sizeof(int));
    if (arr == NULL) {
        printf("Échec de l'allocation de mémoire.\n");
        return 1;
    }
```

```
printf("Entrez les valeurs pour le tableau initial :\n");
for (int i = 0; i < tailleInitiale; i++) {
    scanf("%d", &arr[i]);
}

printf("Tableau initial : ");
afficherTableau(arr, tailleInitiale);

printf("Entrez la nouvelle taille du tableau : ");
scanf("%d", &nouvelleTaille);

arr = (int *)realloc(arr, nouvelleTaille * sizeof(int));
if (arr == NULL) {
    printf("Échec de l'allocation de mémoire.\n");
    return 1;
}

printf("Entrez les valeurs supplémentaires pour le tableau :\n");
for (int i = tailleInitiale; i < nouvelleTaille; i++) {
    scanf("%d", &arr[i]);
}

printf("Tableau redimensionné : ");
afficherTableau(arr, nouvelleTaille);
```

```
    free(arr);



    return 0;

}
```