Travaux Pratiques n°8

Exercice 1

Analyser le programme suivant et donner la suite des affichages produits :

```
#include <stdio.h>

int main (void) {
  int a, b;
  int *p, *q;
  a = 3; b = 4;
  printf("a = %d, adresse de a = %p, b = %d; adresse de b = %p\n", a, &a, b, &b);
  p = &a; q = &b;
  printf("p = %p, valeur pointee par p = %d, q = %p, valeur pointee par q = %d\n", p, *p, q, *q);
  *p += 1; *q += 1;
  printf("a = %d, b = %d\n", a, b);
  p = q;
  printf("p = %p, valeur pointee par p = %d, q = %p, valeur pointee par q = %d\n", p, *p, q, *q);
  *p += 10;
  printf("a = %d, b = %d\n", a, b);
  return 0;
}
```

Exercice 2

Dans cet exercice, on se propose d'écrire pas-à-pas un programme. Tout doit être fait dans la fonction principale.

- Déclarer un entier **i** et un pointeur sur un entier **p**.
- Affecter la valeur 5 à **i**.
- Afficher la valeur de **i**, et celle de l'adresse de **i**.
- Faire pointer **p** à l'adresse de **i**.
- Que contient alors l'expression ***p** ? Vérifier en affichant la valeur de cette expression.
- Déclarer un nouveau pointeur sur entier **q** et le faire pointer à la même adresse que **p**.
- Afficher les expressions du programme contenant la valeur 5, ainsi que leur adresse respective.
- Augmenter de 2 façons différentes la valeur de **i** de 2 sans passer par l'intermédiaire de **i**.

Exercice 3

Si i et j sont des entiers et p et q, des pointeurs sur des entiers, donner le résultat des instructions suivantes :

1) p = &i ;
2) p = &∗&i ;
3) i = ∗&∗&i ;
4) ∗p = &j;
5) i = (**int**)p ;
6) q = &p ;

Exercice 4

Trouver l'erreur contenue dans les extraits de programmes suivants :

1)
```
int *p ;
*p = 5 ;
```

2)
```
int x = 5 ;
int *p = &x ;
p = 9 ;
```

Exercice 5

Comparer les 3 fonctions **permute1**, **permute2**, **permute3** et donner les affichages produits :

```
#include <stdio.h>
void permute1 (int a, int b) {
  int tmp;
  tmp = a;
  a = b;
  b = tmp;
}

void permute2 (int a, int *b) {
  int tmp;
  tmp = a;
  a = *b;
  *b = tmp;
}

void permute3 (int *a, int *b) {
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}

int main (void) {
  int n = 2, m = 4;
  printf("n = %d, m = %d\n", n, m);
  permute1 (n, m);
  printf("n = %d, m = %d\n", n, m);
  n = 2, m = 4;
  permute2 (n, &m);
  printf("n = %d, m = %d\n", n, m);
  n = 2, m = 4;
  permute3 (&n, &m);
  printf("n = %d, m = %d\n", n, m);
  return 0;
}
```

Exercice 6

Ecrire de 2 manières différentes une fonction qui calcule le carré d'un nombre en utilisant un passage par valeur puis un passage par adresse. L'affichage se fera, dans les 2 cas dans le programme principal, au retour de la fonction.

Exercice 7

Ecrire une fonction incremente qui prend 3 paramètres en entrée (**a**, **b** et **i**), et qui ajoute à **a** et **b** la valeur **i**. Cette fonction retourne la valeur 1 si **i** vaut 0, 0 sinon.

Exercice 8

Ecrire le code C permettant de créer un tableau **tab** de 17 entiers en utilisant un pointeur et la fonction malloc. Libérer ensuite la mémoire allouée au tableau avec la fonction free.

Exercice 9

Ecrire le code C permettant de créer un tableau **tab** à deux dimensions de 15 lignes et 17 colonnes entiers en utilisant un pointeur et la fonction malloc. Libérer ensuite la mémoire allouée au tableau avec la fonction free.

Exercice 10

- Ecrire une fonction **saisir_vect(n)** permettant de créer un vecteur dont la taille **n** et les valeurs sont entrées à la main. Cette fonction retourne le vecteur ainsi créé.
- Ecrire une fonction **afficher_vect(v, n)** permettant d'afficher l'ensemble des éléments d'un vecteur **v** de taille **n**.
- Ecrire une fonction qui calcule la moyenne des éléments du vecteur.

Exercice 11

- Ecrire une fonction **saisir_mat(l,c)** qui demande à l'utilisateur les nombres de lignes **l** et de colonnes **c** d'une matrices d'entiers, puis remplit la matrice en demandant les valeurs à l'utilisateur. Cette fonction retourne la matrice ainsi créée.
- Ecrire une fonction **affiche_mat(m,l,c)** qui permet d'afficher une matrice **m** de **l** lignes et **c** colonnes.

Exercice 12

- Ecrire une fonction permettant de multiplier une matrice par un entier.
- Ecrire une fonction permettant d'additionner deux matrices. On n'oubliera pas de traiter les cas d'erreur.
- Ecrire une fonction permettant de multiplier deux matrices. On n'oubliera pas de traiter les cas d'erreur.