Programmation en C

Atelier 03 : Pointeurs, Tableaux, Chaînes de caractère

Pour mener à bien cet atelier, chaque étudiant devra créer un projet par exercice en suivant la notation : *Atl03_ExoX_Prenom_Nom*.

Le *X* doit être remplacé le numéro correspondant de l'exercice. *Prenom* et *Nom* doit être remplacé par votre prenom et nom sans les accents.

Exercice 1

Soit le morceau de programme suivant :

```
int a;
int *pa;
double x;
int **p;
a=8;
pa=&a;
x=3.14159;
p=&pa;
**p=281;
```

En supposant que la mémoire soit structurée en octets, que les entiers soient codés sur 4 octets, les pointeurs sur 4 octets et que la zone de mémoire automatique soit située en début d'exécution à l'adresse 2004, représentez la mémoire finale de ce programme.

Exercice 2

Soit P **un pointeur** qui "**pointe**" sur un **tableau** A:

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
   int *P;
   P = A;
```

Quelles valeurs ou **adresses** fournissent ces expressions:

a. *P+2
b. *(P+2)
c. &A[4]-3
d. A+3
e. &A[7]-P
f. P+(*P-10)
g. *(P+*(P+8)-A[7])

Soit P un pointeur qui 'pointe' sur un tableau A:

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions:

a) `*P+2` => la valeur 14
b) `*(P+2)` => la valeur 34
c) `&P+1` => l'adresse du pointeur derrière le pointeur P (rarement utilisée)
d) `&A[4]-3` => l'adresse de la composante A[1]
e) `A+3` => l'adresse de la composante A[3]
f) `&A[7]-P` => la valeur (indice) 7
g) `P+(*P-10)` => l'adresse de la composante A[2]
h) `*(P+*(P+8)-A[7])` => la valeur 23

Exercice 3

Créez une fonction `sommeTableau` qui renvoie la somme des valeurs contenues dans le tableau (utilisez un `return` pour renvoyer la valeur). Ecrire le programme principal.

Prototype de la fonction à créer :

```
int sommeTableau(int tableau[], int tailleTableau);
```

Exercice 4

Créez une fonction `moyenneTableau` qui calcule et renvoie la moyenne des valeurs. Ecrire le programme principal.

Prototype :

```
double moyenneTableau(int tableau[], int tailleTableau);
```

Exercice 5

Créez une fonction `copierTableau` qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau. Ecrire le programme principal.

Prototype :

```
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

Exercice 6

Créez une fonction `maximumTableau` qui aura pour rôle de remettre à 0 toutes les cases du tableau ayant une valeur supérieure à un maximum. Cette fonction prendra en paramètres le tableau ainsi que le nombre maximum autorisé (`valeurMax`). Toutes les cases qui contiennent un nombre supérieur à `valeurMax` doivent être mises à 0. Ecrire le programme principal.

Prototype

```
void maximumTableau(int tableau[], int tailleTableau, int valeurMax);
```

Exercice 7

Créez une fonction `ordonnerTableau` qui classe les valeurs d'un tableau dans l'ordre croissant. Ainsi, un tableau qui vaut `{15, 81, 22, 13}` doit à la fin de la fonction valoir `{13, 15, 22, 81}`. Ecrire le programme principal.

Prototype :

```
void ordonnerTableau(int tableau[], int tailleTableau);
```

Exercice 8

Écrivez la fonction qui retourne l'indice du premier élément strictement négatif parmi les n premiers éléments d'un tableau de `double` (`-1`si aucun élément n'est négatif). Ecrire le programme principal.

Prototype :

```
int indice_premier_negatif (double t[], int n);
```

Exercice 9

Écrivez la fonction qui retourne la valeur du plus grand des n premiers éléments d'un tableau de `double`. Ecrire le programme principal.

Prototype :

```
double valeur_plus_grand (double t[], int n);
```

Exercice 10

Écrivez la fonction qui retourne l'indice du plus grand des n premiers éléments d'un tableau de `double` (en cas d'ex-æquo, l'indice du premier d'entre eux). Ecrire le programme principal.

Prototype :

```
int indice_plus_grand (double t[], int n);
```

Exercice 11

A partir de deux tableaux saisis, écrivez une fonction qui calcule le schtroumpf des deux tableaux. Ecrire le programme principal.

Pour calculer le schtroumpf, il faut multiplier chaque élément du tableau 1 par chaque élément du tableau 2, et additionner le tout. Par exemple si l'on a :

Tableau 1 :

4	8	7	12

Tableau 2 :

3	6

Le Schtroumpf sera :

3 * 4 + 3 * 8 + 3 * 7 + 3 * 12 + 6 * 4 + 6 * 8 + 6 * 7 + 6 * 12 = 279

Exercice 12

Ecrire une fonction somme qui permet de faire l'addition de deux nombres complexes

Ecrire le programme principal qui :

- saisie les parties réelles et les parties imaginaires de deux nombres complexes,
- calcule la somme de deux nombres complexes (fait appel à la fonction somme),
- affiche le résultat de la somme

Exercice 13

Ecrire une fonction SaisieTableau qui permet la saisie un tableau Tab d'entier de dimension N.

Ecrire une fonction AfficheTableau qui affiche le tableau Tab.

Ecrire le programme principal.

Ajouter au programme une fonction InverseTableau qui inverse le tableau Tab sans utiliser de tableau d'aide (la fonction InverseTableau doit échanger les éléments du tableau à l'aide de deux indices qui parcourent le tableau en commençant respectivement au début et à la fin du tableau et qui se rencontrent en son milieu).

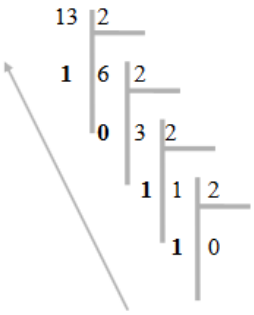
Exercice 14

On veut convertir un nombre décimal en binaire selon la méthode suivante:

1. Tant que le nombre décimal est différent de zéro, on le divise par deux et on met le reste de la division dans un tableau.

2. On inverse les éléments du tableau.

Exemple: le nombre 13 vaut 1101 en binaire



1. Ecrire une fonction inverser qui permet d'inverser les éléments d'un tableau.
2. Ecrire une fonction convertir qui permet de convertir un nombre décimal en binaire.
3. Ecrire une fonction affiche qui affiche les éléments d'un tableau.
4. Ecrire le programme principal.

Les prototypes des fonctions:

```
void inverser(int *tab, int nb);
void convertir (int dec, int *bin, int *nb);
void affiche (int *tab, int nb);
```

Exercice 15

- Ecrire une fonction **saisir** qui permet de saisir une matrice carrée
- Ecrire une fonction **afficher** qui permet d'afficher les éléments d'une matrice
- Ecrire une fonction **additionner** qui permet d'additionner deux matrices
- Ecrire une fonction **multiplier** qui permet de multiplier deux matrices
- Ecrire une fonction **menu** qui demande à l'utilisateur quelle est l'opération à effectuer

Ecrire le programme principal

Tableau à deux dimensions

1. Déclaration et initialisation d'un tableau à deux dimensions

La déclaration d'un tableau à deux dimensions se fait ainsi :

```
TYPE nomTableau[n][m]
```

Où :

- TYPE représente le type des éléments du tableau ;
- nomTableau est l'identificateur de la variable de type tableau ;
- n et m sont les dimensions du tableau.

Exemple :

```
float T[3][4] ;
```

déclare un tableau de 3 x 4 éléments de type réel, notés :

T[0][0]	T[0][1]	T[0][2]	T[0][3]
T[1][0]	T[1][1]	T[1][2]	T[1][3]
T[2][0]	T[2][1]	T[2][2]	T[2][3]

Comme pour les tableaux à un indice, il est possible d'initialiser par défaut un tableau à deux indices.

Exemple : Les deux déclarations suivantes sont équivalentes :

```
int T[3][4] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } } ;
int T[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 } ;
```

2. Utilisation d'un tableau à deux dimensions

Les tableaux à deux indices s'utilisent comme les variables simples ou comme les tableaux à un indice :

```
nomTableau[indice1][indice2]
```

où indice1 et indice 2 sont les indices, c'est-à-dire des expressions à valeurs entières comprises respectivement entre 0 et n-1 pour le premier indice et entre 0 et m-1 pour le deuxième.

3. Utilisation d'un tableau dans une fonction

Comme pour un tableau à un indice, pour passer en paramètre un tableau, on donne le pointeur sur le tableau ainsi que son nombre de colonnes et de lignes.

Exemple : Déclaration de la fonction d'affichage d'un tableau

```
void afficherTableau(int T[3][4], int nbColonnes, int nbLignes)
```

Il faut noter que *T* est défini avec sa taille (allouée en mémoire).