TD8 : Tableaux à deux dimensions (Corrigé)

Exercice 1 : Manipulations de tableaux*

Soit un tableau à deux dimensions de 5 par 5 entiers complètement initialisé :

```
6 2 3 5 6
4 6 2 6 1
1 3 6 7 9
1 6 3 6 8
6 0 1 4 6
```

Question 1.1 : Effectuer les actions suivantes :
- afficher le tableau
- afficher le tableau constitué uniquement des lignes d'indice pair
- afficher le tableau constitué uniquement des éléments d'indice impair de chaque ligne

Correction :

```
1  int tab[5][5]={{6,2,3,5,6},{4,6,2,6,1},{1,3,6,7,9},{1,6,3,6,8},{6,0,1,4,6}};
2  int i,j;
3  for (i=0; i<5; i++)
4  {
5    for (j=0; j<5; j++)
6       printf ("%d ", tab[i][j]);
7     printf ("\n");
8  }
9
10 for (i=0; i<5; i+=2)
11 {
12   for (j=0; j<5; j++)
13      printf ("%d ", tab[i][j]);
14    printf ("\n");
15 }
16
17 for (i=0; i<5; i++)
18 {
19   for (j=1; j<5; j+=2)
20      printf ("%d ", tab[i][j]);
21    printf ("\n");
22 }
```

◇

Question 1.2 : Écrire un algorithme qui permet premièrement d'afficher la diagonale (de gauche à droite), puis la 2e diagonale

Correction :

```
1  int tab[5][5]={{6,2,3,5,6},{4,6,2,6,1},{1,3,6,7,9},{1,6,3,6,8},{6,0,1,4,6}};
2  int i=0 ;
3  while (i< 5)
4  {
5     printf ("%d", tab[i][i]);
6    i++;
7  }
```

◇

Correction :

```
int tab[5][5]={{6,2,3,5,6},{4,6,2,6,1},{1,3,6,7,9},{1,6,3,6,8},{6,0,1,4,6}};

int i=0 ;
int k=4;
while (i < 5)
{
   printf ("%d", tab[i][k]);
  i++;
  k--;
}
```

◇

Question 1.3 : Tester si les nombres qui apparaissent dans les deux diagonales sont tous égaux à une seule et même valeur. Le programme affichera "OUI" ou "NON" en fonction du résultat.
Correction :

```
int tab[5][5]={{6,2,3,5,6},{4,6,2,6,1},{1,3,6,7,9},{1,6,3,6,8},{6,0,1,4,6}};
int val = tab[0][0];

int i=0 ;
int k=4;

while ((i < 5) && (tab[i][i] == val) && (tab[i][k] == val)) // garder l'ordre des conditions
{
  i++;
  k++;
}
if (i==5)
   printf ("OUI");
else
   printf ("NON");
```

◇

Exercice 2 : Échange de triangles**

Écrire l'algorithme qui échange le triangle inférieur avec le triangle supérieur dans un tableau à deux dimensions. C'est donc le tableau obtenu en faisant une symétrie par rapport à la diagonale principale.

Exemple :

```
10 11 45 78                    10 23 56 47
23 44 12 56     ---\           11 44 90 78
56 90 67 89     ---/           45 12 67 55
47 78 55 34                    78 56 89 34
```

Correction :

```
int tab[4][4]={{10, 11, 45, 78},{23, 44, 12, 56}, {56, 90, 67, 89},{47, 78, 55, 34}};
int nbLigEff=4;
int nbColEff=4;

int temp;
int i=0;
while (i<nbLigEff)
```

```
8  {
9    j=i+1;
10   while (j<nbColEff)
11   {
12     temp = tab[i][j];
13     tab[i][j] = tab[j][i];
14     tab[j][i] = temp;
15     j=j+1;
16   }
17   i=i+1;
18 }
```

Exercice 3 : Allocation dynamique**

Refaire les exercices précédant en déclarant à chaque fois des tableaux 2d dynamiques, où la saisie de la taille et des éléments du tableau se fait par l'utilisateur.

TP8 : Tableaux à deux dimensions (Corrigé)

Exercice 4 : Manipulation des matrices en dynamique**

Question 4.1 : Écrire un programme qui permet de construire une matrice nulle de taille $lignes \times colonnes$.

Correction :

```
int nblignes;
int nbcolones;

 printf ("combien de lignes : ");
scanf("%d", &nbligne);

ntf("combien de colonnes : ");
scanf("%d", &nbcolones);

I = (int**) malloc(nblignes*sizeof(int*));

int i;
int j ;
for (i=0 ; i<nblignes ; i++)
{
  I[i]=(int*) malloc(nbcolonnes*sizeof(int));

}
for (i=0; i<nblignes; i++)
{
  for (j=0; j<nbcolonnes; j++)
    I[i][j]=0;
}
```

Question 4.2 : Modifier le programme pour construire la matrice carrée identité I (tous les éléments de la diagonale sont égaux à 1, les autres valent 0).

Correction :

Demander seulement une dimension (matrice carrée). Même initialisation, puis

```
for (i=0; i<nblignes; i++)
{
  I[i][i]=1;
}
```

Question 4.3 : Écrire une fonction qui calcule la trace d'une matrice carrée, c'est-à-dire la somme des coefficients diagonaux de cette matrice.

Correction :

```
int trace(int **A, int nblignes, int nbcolonnes)
{
  if (nblignes != nbcolonnes)
  {
     printf ("Le calcul est impossible");
    return -1;
  }
```

```
  else
  {
    int tr=0;
    for (i=0; i<nblignes; i++)
      tr=tr+A[i][i];

    return tr;
  }
}
```

Question 4.4 : Écrire une fonction qui teste si un élément x appartient à une matrice A.
Correction :

```
int appartenance(int **A, int nblignes, int nbcolonnes, int x)
{
  int i;
  int j;
  for (i=0; i<nblignes; i++)
  {
    for (j=0; j<nbcolonnes; j++)
      if (A[i][j] == x)
        return 1;
  }
  return 0;
}
```

Question 4.5 : Écrire une fonction qui permute deux colonnes d'une matrice.
Correction :

```
void permutationColonnes(int **A, int nblignes, int k, int l)
{
  int i=0;
  int tempo;
  while (i<nblignes)
  {
    tempo=A[i][k];
    A[i][k] = A[i][l];
    A[i][l] = tempo;
    i++;
  }
}
```

Question 4.6 : Écrire une fonction qui permute deux lignes d'une matrice.
Correction :

```
void permutationLignes(int **A, int k, int l)
{
        int *tempo;
        tempo=A[k];
        A[k] = A[l];
        A[l] = tempo;
}
```

Question 4.7 : Écrire une fonction qui remplace dans une matrice A toutes les occurrences de x par y.
Correction :

```
int remplace(int **A, int nblignes, int nbcolonnes, int x, int y)
{
  int i, j;
  for (i=0, i<nblignes, i++)
  {
    for (j=0, j<nbcolonnes, j++)
    {
      if (A[i][j] == x)
        A[i][j] = y;
    }
  }
}
```

Exercice 5 : Addition de matrices***

Le but de cet exercice est de permettre la multiplication de matrice de flottants, ainsi que la lecture et l'écriture de matrices dans des fichiers.

Question 5.1 : Définir la fonction `allouеMatrice` prenant en paramètre un nombre de lignes et un nombre de colonnes et retournant une matrice de flottants allouée dynamiquement.
Correction :

```
float ** allouеMatrice(int n, int m)
{
  float ** mat = malloc(sizeof(float*) * n);
  int i;
  for (i = 0 ; i < n ; i++)
    mat[i] = malloc(sizeof(float) * m);
  return mat;
}
```

Question 5.2 : Définir la fonction `libereMatrice` prenant en paramètre une matrice allouée dynamiquement ainsi que son nombre de lignes et libérant la mémoire prise par cette matrice.
Correction :

```
void libereMatrice(float **mat,int n)
{
  int i;
  for (i = 0 ; i < n ; i++)
    free(mat[i]);
  free(mat);
}
```

Question 5.3 : Définir la fonction `lireMatrice` prenant en paramètre deux pointeurs d'entiers et un nom de fichier. Cette fonction ouvrira le fichier dont le nom est passé en paramètre. Ce fichier doit contenir le nombre de lignes, le nombre de colonnes ainsi que les valeurs d'une matrice. La fonction utilisera ces informations pour allouer dynamiquement une matrice et l'initialiser avec les valeurs du fichier. La fonction modifiera les deux pointeurs d'entiers pour qu'ils contiennent respectivement le nombre de lignes et de colonnes de la matrice et retournera l'adresse mémoire de la matrice (ou `NULL` en cas d'erreur).
Correction :

```c
float ** lireMatrice(int *n, int *m, char nomF[])
{
  FILE * f = fopen(nomF,"rt");
  if (f==NULL)
    return NULL;

  //Lecture du nombre de lignes et de colonnes de la matrice
  fscanf(f,"%d",n);
  fscanf(f,"%d",m);

  //Allocation de la matrice
  float ** matrice = alloueMatrice(*n,*m);
  if (matrice==NULL)
    return NULL;

  //On lit les valeurs
  int i,j;
  for (i = 0 ; i < *n ; i++)
    for (j = 0 ; j < *m ; j++)
      fscanf(f,"%f",&matrice[i][j]);

  if (fclose(f))//En cas de problème lors de la fermeture
  {
    libereMatrice (matrice,*m);//On libère la mémoire
    return NULL;
  }
  return matrice;
}
```

Question 5.4 : Définir la fonction `ecrireMatrice` prenant en paramètre une matrice, son nombre de lignes et de colonnes et le nom d'un fichier et écrivant dans ce fichier les informations de la matrice.
Correction :

```c
int ecrireMatrice(float ** mat, int n, int m, char nomF[])
{
  FILE * f = fopen(nomF,"wt");
  if (f==NULL)
    return 0;

  fprintf (f,"%d \n %d \n",n,m);
  int i,j;
  for (i = 0 ; i < n ; i++)
  {
    for (j = 0 ; j < m ; j++)
      fprintf (f,"%6f",mat[i][j]);
    fprintf (f,"\n");
  }
  return !fclose(f);
}
```

Question 5.5 : Écrire une fonction `addition_mat` qui permet de faire l'addition de deux matrices de même dimension.
Correction :

```
float **addition_mat(float **mat1, float **mat2, int nblignes, int nbcolonnes)
{
  float ** mat3 = allouеMatrice(nblignes,nbcolonnes);
  int i,j;
  for (i=0 ; i<nblignes ; i++)
  {
    for (j=0 ; j<nbcolonnes ; j++)
      mat3[i][j] = mat1[i][j] + mat2[i][j];
  }
  return mat3;
}
```

◇

Question 5.6 : Écrire un programme principal permettant de créer deux matrices à partir des fichiers de données `mat1.txt` et `mat2.txt`, de calculer la somme de ces deux matrices et d'écrire ce résultat dans le fichier `mat3.txt`. Les 3 matrices seront également affichées à l'écran.
Correction :

```
int main()
{
  int n1,m1;
  float ** mat1 = lireMatrice(&n1,&m1,"m1.txt");
   printf ("Premiere matrice : \n");
   afficheMatrice (mat1,n1,m1);
   printf ("\n\n");

  int n2,m2;
  float ** mat2 = lireMatrice(&n2,&m2,"m2.txt");
   printf ("Deuxieme matrice : \n");
   afficheMatrice (mat2,n2,m2);
   printf ("\n\n");

  if (n1==n2 && m1==m2)//Si les matrices ont les mêmes dimensions
  {
    float **mat3 = addition_mat(mat1,mat2,n1,m1);
     printf ("Somme des deux matrices :\n");
     afficheMatrice (mat3,n1,m1);
     ecrireMatrice (mat3,n1,m1,"m3.txt");
     libereMatrice (mat3,n1);
    mat3 = NULL;
  }
   libereMatrice (mat1,n1);
  mat1 = NULL;
   libereMatrice (mat2,n2);
  mat2 = NULL;
  return 0;
}
```

◇

Exercice 6 : Carré magique***

Un carré (tableau d'entiers de N lignes et N colonnes initialisées) est dit magique lorsque la somme d'une ligne, d'une colonne ou d'une diagonale quelconque est toujours égale au même nombre. Voici un exemple de carré magique :

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

Question 6.1 : Concevoir une fonction créant une matrice carrée dont les coefficients sont des entiers positif saisi par l'utilisateur.

Question 6.2 : Concevoir une fonction qui vérifie si un carré est magique.

Correction :

```
1  int carre_magique(int **tabMag, int nblignes)
2  {
3    int OK = 1;
4    int i;
5    int j;
6    int somme = 0;
7    int sommeSuiv = 0;
8    int  taille  = nblignes;
9
10   //initialisation de la somme pour la première ligne
11   for (j=0; j< taille ; j++)
12     somme += tabMag[0][j];
13
14   //comparaison avec les autres lignes
15   i= 1;
16   while (OK && i < taille)
17   {
18     sommeSuiv = 0;
19     for (j=0 j< taille ; j++)
20       sommeSuiv += tabMag[i][j];
21
22     if (sommeSuiv != somme)
23       OK = 0;
24     i++;
25   }
26
27   //  comparaison avec les colonnes
28   j = 0;
29
30   while (OK && j < taille)
31   {
32     sommeSuiv = 0;
33     for (i=0; i<  taille ; i++)
34       sommeSuiv += tabMag[i][j];
35
36     if (sommeSuiv != somme)
37       OK = 0;
38     j++;
39   }
40
41   // comparaison avec les deux diagonales
42   // la premiere
43   if (OK)
44   {
45     sommeSuiv = 0;
46     for (i=0; i<  taille ; i++)
47       sommeSuiv += tabMag[i][i];
48
```

```
49      if (sommeSuiv != somme)
50        OK = 0;
51    }
52
53    //la seconde
54    if (OK)
55    {
56      sommeSuiv = 0;
57      for (i=0; i< taille ; i++)
58        sommeSuiv += tabMag[i][taille -1 -i];
59
60      if (sommeSuiv != somme)
61        OK = 0;
62
63      return OK;
64    }
65  }
```