

LIFAP1 – TD 7 : Tableaux à deux dimensions

Objectifs : Apprendre à manipuler les tableaux à deux dimensions et approfondir les notions algorithmiques vues jusqu'à présent (condition, boucles, ...).

1. Soit T un tableau 2D carré de taille 5*5 contenant des entiers. Écrire la déclaration et l'initialisation à 0 d'une telle structure de données.

```
Déclaration : T : tableau[5][5] d'entiers
Procédure InitTab(T : tableau[5][5] d'entiers)
Donnée / Résultat : T
Variable locales : i,j : entier
Début
  Pour i allant de 0 à 4 par pas de 1 faire
    Pour j allant de 0 à 4 par pas de 1 faire
      T[i][j] ← 0
    Fin Pour
  Fin Pour
Fin InitTab
```

2. Écrire un sous-programme RemplirTab qui propose à l'utilisateur de remplir un tableau T de taille 5*5.

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

```
Procédure RemplirTab(T : tableau[5][5] d'entiers)
Donnée / Résultat : T
Variable locales : i,j : entier
Début
  Pour i allant de 0 à 4 par pas de 1 faire
    Pour j allant de 0 à 4 par pas de 1 faire
      Afficher (« donnez la valeur »)
      Saisir(T[i][j])
    Fin Pour
  Fin Pour
Fin remplirTab
```

3. Écrire deux procédures d'affichage d'un tableau 2D de taille 5*5
 - a. Affichage_2D_ligne : qui affichera le tableau ligne par ligne

```
Procédure Affiche_2D_ligne(T : tableau[5][5] d'entiers)
Donnée / Résultat : T
Variable locales : i,j : entier
Début
  Pour i allant de 0 à 4 par pas de 1 faire // ligne
    Pour j allant de 0 à 4 par pas de 1 faire // colonne
      Afficher (T[i][j])
    Fin Pour // fin colonne
    Afficher (saut de ligne)
  Fin Pour
Fin Affiche_2D_ligne
```

b. Affichage_2D_colonne : qui affichera le tableau colonne par colonne

```
Procédure Affiche_2D_colonne(T : tableau[5][5] d'entiers)
Donnée / Résultat : T
Variable locales : i,j : entier
Debut
  Pour i allant de 0 à 4 par pas de 1 faire      // colonne
    Pour j allant de 0 à 4 par pas de 1 faire    // ligne
      Afficher (T[j][i])
    Fin Pour                                     // fin ligne
  Afficher (saut de ligne)
Fin Pour
Fin Affiche_2D_colonne
```

4. Écrire trois fonctions permettant sur un tableau 2D de taille 5*5:

a. de calculer la somme des éléments d'une ligne (le numéro de la ligne étant passé en paramètre)

```
Fonction SommeLigne(T : tableau[5][5] d'entiers, ligne : entier) : entier
Donnée / Résultat: T
Donnée : ligne : numéro de la ligne dont on veut calculer la somme
Résultats : somme des éléments de la ligne "ligne"
Variables locales : i,som_lig : entier
Debut
  som_lig ← 0
  Pour i allant de 0 à 4 par pas de 1 faire
    som_lig ← som_lig + T[ligne][i]
  Fin Pour
  Retourner (som_lig)
Fin SommeLigne
```

b. de calculer la somme des éléments d'une colonne (le numéro de la colonne étant passé en paramètre)

```
Fonction SommeColonne(T : tableau[5][5] d'entiers, colonne : entier) : entier
Donnée / Résultat: T
Donnée : colonne : numéro de la colonne dont on veut calculer la somme
Résultats : somme des éléments de la colonne "colonne"
Variables locales : i,som_col : entier
Debut
  som_col ← 0
  Pour i allant de 0 à 4 par pas de 1 faire
    som_col ← som_col + T[i][colonne]
  Fin Pour
  Retourner (som_col)
Fin SommeColonne
```

c. de calculer les sommes des éléments de chaque diagonale (dans la mesure où le tableau est bien carré)

```
Fonction SommeDiagonale(T : tableau[5][5] d'entiers, som_diag2 : entier) : entier
Précondition : le tableau T est carré
Donnée / Résultat: T, som_diag2
Résultats : somme des éléments de la première diagonale
Variable locales : i,som_diag1 : entier
Debut
  som_diag ← 0
  som_diag2 ← 0
  Pour i allant de 0 à 4 par pas de 1 faire
    som_diag1 ← som_diag1 + T[i][i]
```

```

    som_diag2 ← som_diag2 + T[i][j-i]

    Fin Pour
    Retourner (som_diag1)
Fin

```

5. Écrire un sous-programme `RecherchePlusGrand` permettant de rechercher le plus grand élément de ce tableau et de retourner l'indice de ligne et l'indice de colonne correspondant à cet élément ainsi que l'élément lui-même.

```

Fonction RecherchePlusGrand(T : tableau[5][5] d'entiers, lig_max : entier, col_max :
entier) : entier
Donnée / Résultat: T, lig_max et col_max
Résultat : valeur du plus grand élément de T
Variable locales : i,j,maxi : entier
Debut
    maxi ← T[0][0]
    lig_max ← 0
    col_max ← 0
    Pour i allant de 0 à 4 par pas de 1 faire
        Pour j allant de 0 à 4 par pas de 1 faire
            Si (maxi < T[i][j]) alors
                maxi ← T[i][j]
                lig_max ← i
                col_max ← j
            Fin Si
        Fin Pour
    Fin Pour
    Retourner maxi
Fin RecherchePlusGrand

```

6. Soit T un tableau à 2 dimensions de taille M * N contenant des entiers. Ce tableau est rempli avec des nombres sur les L premières lignes et les C premières colonnes. Écrire en langage algorithmique un sous-programme permettant de remplir un tableau 1D avec la somme des colonnes de T. Attention à ne bien parcourir que les colonnes et les lignes remplies.

1	5	6	4	
8	9	0	6	
3	2	7	1	
12	16	13	11	

On commence par définir 2 constantes :

```

M = 10
N = 10

```

```

Procédure Remplir_Somme_Colonne(T : tableau[M][N] d'entiers, Tab_Res[N],
Taille_L : entier, Taille_C : entier)
Précondition : Tab_Res initialisé à 0
Donnée / Résultat : T, Tab_Res
Données : Taille_L, Taille_C
Variable locales : i,j : entier
Debut
    Pour i allant de 0 à Taille_L-1 par pas de 1 faire
        Pour j allant de 0 à Taille_C-1 par pas de 1 faire

```

```

        Tab_Res[i]=Tab_Res[i] + T[i][j]
    Fin Pour
Fin Pour
Fin Remplir_Somme_Colonne

```

7. Écrire un sous-programme `TrianglePascal` permettant de remplir les cases d'un tableau 2D avec les coefficients du triangle de Pascal.

Remarque : La hauteur du triangle sera passée en paramètre et on supposera la fonction combinaison écrite.

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

```

Procédure TrianglePascal(T : tableau[TAILLE][ TAILLE] d'entiers, hauteur : entier)
Précondition : hauteur <= TAILLE
Donnée / Résultat : T
Donnée : hauteur : nombre de lignes du triangle à calculer
Variable locales : i,j : entier
Début
    Pour i allant de 0 à hauteur-1 par pas de 1 faire
        Pour j allant de 0 à i par pas de 1 faire
            T[i][j] ← combinaison(i,j)
        Fin Pour
    Fin Pour
Fin trianglePascal

```

Pour s'entraîner

- Écrire une procédure en C qui prend comme paramètre un tableau $n \times n$ et qui met dans chaque case située **sous** la diagonale le produit du numéro de la ligne par le numéro de la colonne comme dans l'exemple suivant (les autres cases n'étant pas modifiées) :
La taille du tableau sera passée en paramètre.

	0	1	2	3	4
0	0				
1	0	1			
2	0	2	4		
3	0	3	6	9	
4	0	4	8	12	16

Version algo :

```

Procédure Diagonale(T : tableau[TAILLE][ TAILLE] d'entiers, hauteur : entier)
Donnée / Résultat : T

```

Donnée : hauteur : nombre de lignes du triangle à calculer
Variable locales : i,j : entier
Début

```
    Pour i allant de 0 à hauteur-1 par pas de 1 faire
        Pour j allant de 0 à i par pas de 1 faire
             $T[i][j] \leftarrow i*j$ 
        Fin Pour
    Fin Pour
Fin trianglePascal
```

Version C :

```
void Diagonale (int T[N][N], int hauteur)
{
    int i, j;
    for (i = 0; i < hauteur; i++)
        for (j= 0; j <= i; j++)
        {
             $T[i][j] = i*j$ ;
        }
}
```