TP 3 : Les tableaux

Objectifs :

- Maîtriser les tableaux à une dimension et à deux dimensions.
- Savoir passer des arguments en ligne de commande.

Tableau à une dimension

1. Rappel algorithmique

Un tableau est caractérisé par trois éléments :

- son nom ;
- son nombre d'éléments ;
- et le type des éléments qu'il contient.

Exemple :

precipitations

66,0	72,6	82,8	81,9	107,2	94,1	63,4	74,9	91,4	94,1	80,3	70,7

Figure 1 : Tableaux des précipitations moyennes à Grenoble.

Le tableau *precipitations* contient *12* éléments de type *réel*. ■

2. Déclaration d'un tableau

En langage C, une variable de type tableau doit être déclarée au même titre que les autres variables dans la partie déclaration. La déclaration d'un tableau se fait ainsi :

```
TYPE nomTableau[n]
```

Où :

- TYPE représente le type des données contenues dans le tableau ;
- nomTableau est le nom de la variable de type tableau ;
- n est le nombre d'éléments du tableau.

Exemple :

```
float precipitations[12] ;
```

déclare un tableau precipitations à 12 éléments de type réel. ■

3. Utilisation d'un tableau

En langage C, l'utilisation d'un tableau se fait ainsi :

```
nomTableau[indice]
```

Où indice peut être :

- une variable simple : precipitations[i]
- une constante : precipitations[2]
- une expression arithmétique : precipitations[2*i]

L'indice doit être une valeur entière **comprise entre 0 et n-1 (n étant la taille du tableau)**.

Exemple : Parcours du tableau precipitations

Algorithme :

```
Pour i allant de 0 à 11 par pas de 1
faire
        Afficher(precipitations[i])
FinPour
```

En langage C :

```
for(i = 0 ; i < 12 ; i = i + 1) {
        printf(''%6.2f\n'', precipitations[i]);
}
```

4. Utilisation des tableaux dans les fonctions paramétrées

En langage C, le nom du désigne un pointeur[1] contenant l'adresse du premier élément du tableau.

Exemple : Fonction d'affichage d'un tableau

Soit une fonction *afficherTableau()* ayant deux paramètres d'entrée :

- *precipitations* : le tableau de réels ;
- *n :* le nombre d'éléments du tableau.

```
void afficherTableau(float precipitations[], int n) ;
```

Ainsi, on passe en paramètre le nom du tableau et son nombre d'éléments (c'est le seul moyen pour que la fonction `afficherTableau` connaisse la taille du tableau).

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

```
afficherTableau(precipitations, 12);
```

5. Initialisation par défaut d'un tableau

Comme toute variable d'un type de base, un tableau peut être initialisé explicitement au moment de sa déclaration.

Exemple : Initialisation du tableau de précipitations

```
float precipitations[12] = {66.0, 72.6, 82.8, 107.2, 94.1, 63.4, 74.9,
91.4, 94.1, 80.3, 70.7};
```

Les valeurs utilisées pour l'initialiser doivent toujours être des constantes. De plus, il est possible d'omettre la dimension du tableau pour ce type de déclaration. L'expression suivante est équivalente :

```
float precipitations[] = {66.0, 72.6, 82.8, 107.2, 94.1, 63.4, 74.9,
91.4, 94.1, 80.3, 70.7};
```

Le compilateur détermine la dimension du tableau en fonction du nombre de valeurs énumérées.

[1] La notion de pointeur sera vue plus loin : un pointeur est une variable contenant une adresse mémoire

Exercice

Exercice 1 : Gestion de notes

Le but de cet exercice est d'implanter un programme de gestion des notes d'un groupe d'étudiants. Ce programme doit faciliter l'impression des statistiques concernant les notes obtenues par les étudiants pour un examen. Pour cet exercice, nous supposons que le nombre d'étudiants est fixe et que les notes doivent appartenir à l'intervalle [0 ; 20]. Etant donné que le programme a pour objectif de fournir des statistiques, nous ne tiendrons pas compte du nom des étudiants ; seules les notes nous intéressent.

Les statistiques attendues sont :

- La note la plus basse ainsi que la note la plus élevée ;
- La moyenne, la variance et l'écart-type ;

Pour rappel (voir bibliothèque standard math.h[2]) :

Variance : $\frac{\sum(x-\bar{x})^2}{n}$

Ecart-Type : $\sqrt{\frac{\sum(x-\bar{x})^2}{n}}$

1. Jeu de tests

Le programme réalisé devra être testé. Voici un exemple de test :

0	1	2	3	4
12,0	13,5	8,5	14,7	6,0

Quelles sont les valeurs attendues pour :

Note la plus basse	
Note la plus élevée	
Moyenne des étudiants	
Variance	
Ecart-type	
Rang dans le tableau de la valeur 13,5	
Rang dans le tableau de la valeur 10,5	

Avant d'aborder les questions suivantes, construire trois autres tests.

2. Partie obligatoire[2]

1. Ecrire une fonction *afficherNotes()* qui affiche à l'écran toutes les notes obtenues à l'examen.

2. Ecrire une fonction *minimumNote()* qui prend en entrée un tableau de notes et sa taille et qui retourne la note la plus basse obtenue à l'examen.

3. Ecrire une fonction *maximumNote()* qui prend en entrée un tableau de notes et sa taille et qui retourne la note la plus élevée obtenue à l'examen.

4. Ecrire une fonction *calculeMoyenne()* qui prend en entrée un tableau de notes et sa taille et qui retourne la moyenne des notes.

5. Ecrire une fonction *calculeVariance()* qui prend en entrée un tableau de notes et sa taille et qui retourne la variance.

6. Ecrire une fonction *calculeEcartType()* qui prend en entrée un tableau de notes et sa taille et qui retourne l'écart-type.

7. Ecrire une fonction *rechercherValeur()* qui prend en entrée un tableau de notes, sa taille et une valeur à rechercher. Cette fonction retourne la position de la valeur dans le tableau, -1 si la valeur n'a pas été trouvée.

3. Partie optionnelle

Le but de cette partie est d'afficher graphiquement la répartition des notes. Cet affichage se fera sous la forme d'histogrammes (horizontal et vertical). Le principe est d'afficher autant de croix qu'il y a de notes par intervalle de deux points.

1. Ecrire une fonction qui permet d'afficher un histogramme horizontal. Le résultat attendu est le suivant pour les notes :
   0 ; 13.5 ; 8.5 ; 13.7 ; 20 ; 12 ; 8.5 ; 17 ; 11 ; 10 ; 9.5 ; 4 ; 14 ; 13.5 ; 12 ; 1 ; 15 ; 10.5 ; 7.5 ; 9.5.

```
[  0 ;  2 ] :   *       *
]  2 ;  4 ] :   *
]  4 ;  6 ] :
]  6 ;  8 ] :   *
]  8 ; 10 ] :   *       *       *       *       *
] 10 ; 12 ] :   *       *       *       *
] 12 ; 14 ] :   *       *       *       *
] 14 ; 16 ] :   *
] 16 ; 18 ] :   *
] 18 ; 20 ] :   *
```

2. Ecrire une fonction qui permet d'afficher un histogramme vertical. Le résultat attendu est le suivant pour les mêmes notes :

```
                                        *
                                        *       *       *
                                        *       *       *
    *                                   *       *       *
    *       *               *           *       *       *       *       *       *
-------------------------------------------------------------------------------------------
[ 0 ;  2] ] 2 ;  4] ] 4 ;  6] ] 6 ;  8] ] 8 ; 10] ]10 ; 12] ]12 ; 14] ]14 ; 16] ]16 ; 18] ]18 ; 20]
```

[2] Pour utiliser la bibliothèque standard math.h penser à ajouter dans le Makefile l'option **–lm** lors de l'édition des liens; la documentation des bibliothèques standard peut être trouvée sur internet

Passage de paramètres

Un programme contient obligatoirement une fonction principale : la fonction `main`. Celle-ci peut disposer d'arguments lui permettant de recueillir des informations venant de l'utilisateur.

La fonction `main` a obligatoirement une des signatures suivantes (selon que l'on souhaite, ou pas, qu'elle dispose d'arguments) :

- `int main(void)`
- `int main(int argc, char *argv[])`

Si la fonction main est de la deuxième forme, les arguments ont la signification suivante :

- argc : le nombre de paramètres fournies par l'utilisateur (entier positif ou nul).
- argv[] : contient les paramètres sous forme de chaînes de caractères. argv[0] contient toujours le nom du programme.

Exemple : Contenu des variables argc et argv

```
machine@esisar>./monProgramme 12 toto 20.5
```

argc	4
argv[0]	./monProgramme
argv[1]	12
argv[2]	toto
argv[3]	20.5

Exemple : Utilisation des variables argc et argv

```
#include <stdio.h>
int main(int argc, char * argv[]) {
    int i;
    printf("Nombre d'arguments : %d\n", argc);
    for(i = argc-1 ; i > 0 ; i--) {
          printf("%s\n", argv[i]);
    }
    return 0;
}
```

Exercice

Exercice 2

Tester le programme donné dans l'exemple précédent. Quels sont les résultats obtenus ?

Tableaux 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).

4. Tableau à *n* dimensions

Il est possible de composer à volonté les déclarateurs de telle sorte que l'on peut obtenir des tableaux de tableaux de tableaux…

Exemple : Un tableau à trois indices déclaré ainsi :

```
double tf[2][3][4] ;
```

- tf[2][3][4] est un double ;
- tf[2][3] est un tableau de 4 double ;
- tf[2] est un tableau de 3 tableaux de 4 double ;
- tf est est un tableau de 2 tableaux de 3 tableaux de 4 double.

Exercice

Exercice 3 : Matrices

Les matrices sont un exemple de tableaux à deux dimensions. Nous souhaitons réaliser un module qui permette de créer et d'initialiser une matrice, d'afficher une matrice, d'additionner deux matrices, de calculer la transposée d'une matrice et le produit de deux matrices.

1. Ecrire un programme qui crée une matrice de taille fixe, qui l'initialise et qui l'affiche.
2. Quels sont les résultats attendus dans les opérations suivantes?

Addition de deux matrices :

$$\begin{pmatrix} 1 & 0 & 4 \\ -1 & 2 & -5 \end{pmatrix} + \begin{pmatrix} 2 & -3 & 1 \\ 5 & 8 & 2 \end{pmatrix} = ?$$

- Quelles sont les contraintes pour réaliser une addition de deux matrices ?
- Quelle est la taille de la matrice résultat ?

Matrice transposée :

$$A = \begin{pmatrix} 1 & 0 & 4 \\ -1 & 2 & -5 \end{pmatrix} \qquad \text{et} \qquad {}^tA = ?$$

- Quelle est la taille de la matrice résultat ?

Produit matriciel :

$$\begin{pmatrix} 1 & 0 & 4 \\ -1 & 2 & -5 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 2 & -1 \\ 3 & 4 \end{pmatrix} = ?$$

- Quelles sont les contraintes pour réaliser un produit de deux matrices ?
- Quelle est la taille de la matrice résultat ?

Pour chacune des questions suivantes, écrire au préalable des tests qui serviront à vérifier la correction des fonctions.

3. Ecrire une fonction *afficherMatrice()* qui affiche une matrice.
4. Ecrire une fonction *additionnerMatrices()* qui calcule la somme de deux matrices.
5. Ecrire une fonction *transposee()* qui calcule la matrice transposée.
6. Ecrire une fonction *produitMatriciel()* qui réalise un produit matriciel.