LIFAP1 – TP4 : Tableaux 1D / 2D

Objectifs : Manipulation des tableaux à 1 et 2 dimensions
Utilisation de tailles variables dans les tableaux

1. Les tableaux à une dimension : manipulations de base
   a. Écrivez une procédure `tabRemplir` qui remplit un tableau de taille `TAILLE` en demandant à l'utilisateur les valeurs. On définira `TAILLE` comme une **constante** au début du programme :

```
// En ALGO
Constante : TAILLE : Entier = 5
Procédure tabRemplir(T : Tableau[TAILLE] d'Entier)
// En C
#include <iostream>
Using namespace std;
const int TAILLE=5;
void tabRemplir( ...
```

   b. Écrivez une procédure `tabAff` qui affiche sur la sortie standard le contenu d'un tableau d'entiers
   c. En C, un tableau ne peut avoir une taille variable : sa taille doit être une constante. Pour pouvoir gérer un tableau de taille quelconque une manière de faire est de définir une grande valeur pour `TAILLE` et d'utiliser une valeur `tailleT` pour indiquer la taille réellement utilisée du tableau :

```
// En ALGO
Constante :
TAILLE : Entier = 100
Proc tabAff(T : donnée Tab[TAILLE] d'Entier ;  tailleT :donnée
Entier)

// En C
const int TAILLE=100;
void tabAff(int T[TAILLE], int tailleT)
{...}
```

   Modifiez les procédures des questions a et b pour prendre en compte cette amélioration.

2. **Tri par comptage** : le tri par comptage consiste pour chaque élément du tableau à compter combien d'éléments sont plus petits que lui ; grâce à ce chiffre on connaît sa position dans le tableau résultat. Soit le tableau initial suivant :

Tableau initial	52	10	1	25	62	3	8	55
Tableau comptage	5	3	0	4	7	1	2	6
Tableau résultat	1	3	8	10	25	52	55	62

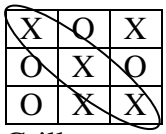
   Écrire l'algorithme d'un sous-programme permettant de trier un tableau de 10 entiers **distincts** en utilisant la méthode décrite précédemment.
   Le **tableau initial** est fourni en paramètre d'entrée, le **tableau de comptage** est calculé dans le sous-programme et permet de remplir et renvoyer le **tableau résultat**.

3. Et maintenant en 2D !!!
   a. Soit un tableau d'entiers à deux dimensions de taille maximum `TAILLE_LIGNE` et `TAILLE_COLONNE`. Écrivez une procédure qui, à partir du nombre de lignes et de colonnes données par l'utilisateur remplit les `tailleL * tailleC` cellules de ce tableau.
   b. Écrivez une procédure qui affiche (proprement) le contenu du tableau précédent.

4. Des caractères dans des tableaux !!!
   Dans cette partie nous allons programmer le jeu du morpion. Pour cela, vous avez besoin d'une grille 3*3 et de 2 joueurs ayant des pions différents (les croix et les ronds).

X	X	O
O	X	X
X	O	O

Grille sans gagnant



Grille avec gagnant

   A tour de rôle, chaque joueur positionne un de ses pions sur la grille. Le jeu se finit quand un joueur a réalisé une ligne, une colonne ou une diagonale avec ses pions (c'est le gagnant) ou quand la grille est pleine (pas de gagnant).
   La grille est représentée par un tableau à 2 dimensions de caractères dont chaque case contiendra soit '_', soit 'O', soit 'X'. Pour réaliser l'implémentation de ce jeu, écrivez les sous-programmes suivants.
   a. Initialisation de la grille du morpion à vide (caractère '_').
      `void initialiseGrille(char grille[3][3])`
   b. Affichage de la grille du morpion : _ indique case vide, O pion joueur 1 et X pion joueur 2.
      `void afficheGrille(char grille[3][3])`
   c. Saisie des coordonnées du nouveau pion à mettre sur la grille. Si les coordonnées sont en dehors de la grille ou si la case possède déjà un pion, la saisie est refusée, un message d'erreur est affiché, et le joueur doit rejouer. Dans le cas où les coordonnées sont correctes, placer le pion sur la grille à cet emplacement.
      `void metUnPionSurLaGrille(char grille[3][3], char &joueur)`
   d. Teste si l'un des joueurs a gagné (ligne, colonne ou diagonale remplie de pions semblables). Dans ce cas, affiche un message pour indiquer le joueur qui a gagné. S'il n'y a pas de gagnant, teste que la grille n'est pas pleine. Si elle est pleine, affiche un message indiquant qu'aucun des joueurs n'a gagné. Retourne TRUE si la grille est pleine ou si un joueur a gagné, FALSE sinon.
      `bool testeFinJeu(char grille[3][3], char joueur)`
   e. Écrivez ensuite le programme principal permettant de dérouler la partie. En voici son algorithme :

```
Algorithme principal :
   Initialisation de la grille à vide
   Tant que (pas de gagnant ou pas grille pleine)
    Afficher grille
    Mettre un pion sur la grille
```