LIFAPI – TP6 : Tableaux 1D

Objectifs : Manipulation de tableaux à 1 dimension

1. Les tableaux à une dimension : taille fixée par une constante
   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 à l'écran le contenu du tableau d'entiers défini précédemment.
   c. Écrivez le programme principal qui teste ces deux sous-programmes.

2. En C/C++, un tableau ne peut pas 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 et testez dans le programme principal avec une taille choisie par l'utilisateur.

3. Tableaux générés aléatoirement et extraction de valeurs caractéristiques.
   Soit T un tableau de taille `TAILLE` rempli jusqu'à `TailleT` (comme dans l'exercice 2). Tous les sous-programmes doivent être testés au fur et à mesure.
   a. Écrivez une procédure `genere_aleatoire` qui remplit les `TailleT` premières valeurs d'un tableau de taille `TAILLE` avec des valeurs aléatoires comprises entre `-15` et `+30`.
   b. Écrivez une procédure `affiche_tab` qui affiche le tableau précédent.
   c. Écrivez un sous-programme `min_max` qui détermine et "retourne" la plus petite valeur **ET** la plus grande valeur du tableau `T`.
   d. Écrivez une fonction booléenne `verifie_tout_positif` qui retourne vrai si tous les éléments du tableau sont strictement positifs, faux sinon.
   e. Écrivez une fonction `compte_superieur` qui compte et retourne le nombre de valeurs supérieures à un entier `n` passé en paramètre dans le tableau `T`.

4. **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**.