# LIFAP1 – TP6 : Tableaux 1D

*Objectifs* : Manipulation de tableaux à 1 dimension

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.h>
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.

```
#include <iostream>
using namespace std ;
// La taille maximale d'un tableau
const int TAILLE=100;

// Demander à l'utilisateur de remplir un tableau ayant tailleT elements
void tabRemplir(int T[TAILLE], int tailleT)
{  int i;
   for (i=0; i<tailleT; i++)
   { cout << "tab["<<i<<"] = ";
     cin >> T[i];
   }
}

// Afficher tous les elements d'un tableau ayant tailleT elements
void tabAff(int T[TAILLE], int tailleT)
{  int i;
   for (i=0; i<tailleT; i++)
   { cout << "Tableau ["<<i<<"] = " << T[i] << endl;
   }
}

int main(void)
{
   int T[TAILLE], tailleT, somme;
   cout << "Entrez la taille du tableau = ";  cin >> tailleT;
   cout << "Remplir le tableau" << endl ;
```

```
    tabRemplir(T,tailleT);
    cout << "Afficher le tableau" << endl ;
    tabAff(T,tailleT) ;
    return 0;
}
```

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**.

```
#include <iostream>
using namespace std;
const int TAILLE = 8 ;


void remplir( int tab[TAILLE])
{   int i;
    for (i=0;i<TAILLE;i++)
    {
        cout<<"Donnez la valeur"<<endl;
        cin>>tab[i];
    }
}
void affiche( int tab[TAILLE])
{
    int i;
    for (i=0;i<TAILLE;i++)
    {
        cout<<tab[i]<<" ";
    }
    cout<<endl;
}

int tri_comptage (int t_init[TAILLE], int t_res[TAILLE])
{
    int i,j;
    int t_nb[TAILLE] ;
    for (i=0;i<TAILLE;i++)
    {
        t_res[i]=0;
        t_nb[i]=0;
        for (j=0;j<TAILLE;j++)
```

```
        {
            if (t_init[j]<t_init[i])
            {
               t_nb[i]++;
            }
        }
    }
   affiche (t_nb);


    for (i=0;i<TAILLE;i++)
    {
        j=t_nb[i];
        t_res[j]=t_init[i];
    }
}

int main (void)
{
    int init[TAILLE], res[TAILLE];

    remplir (init);
    affiche(init);

    tri_comptage (init,res);
    affiche(res);

    return 0;
}
```