







# TD07 – Templates

## Exercice 1 Tableau générique




### 1.1 Première phase → simplement un tableau

-  Concevez une classe de gestion d'un tableau statique de 3 réels.
  -  Cette classe aura une méthode **begin()** retournant le pointeur vers le premier élément et une méthode **end()** retournant le pointeur vers l'élément suivant le dernier.
-  Créez un programme de test initialisant un tableau avec les données { 1, 2, 3 }, puis les affichant dans la console.
  -  L'affichage devra se faire par la syntaxe **cout<<tableau**.


### 1.2 Deuxième phase → dimension configurable à la compilation

-  Modifiez le code précédent de façon à ce que la taille du tableau puisse être choisie dans le code, mais toujours réalisée à l'aide d'une allocation statique.
  -  Réécrire le programme de test pour un tableau de 5 éléments

### 1.3 Troisième phase → type d'éléments configurable

-  Modifiez le code précédent de façon à ce que le type des éléments du tableau puisse être choisi dans le code.
  -  Réécrire le programme de test avec un tableau de 3 éléments réels égaux à { 1.1, 2.2, 3.3 } et un tableau de 5 éléments entiers égaux à {1, 2, 3, 4, 5}.
  -  Pour encore plus de généricité du code, incluez dans la classe **CTableau** des définitions de types **iterator** et **const\_iterator**, alias respectifs de pointeur vers éléments et pointeur vers élément constant.

### 1.4 Quatrième phase → conversion de tableaux

-  Faites en sorte qu'un tableau d'entier puisse être affecté à un tableau de réels (et inversement). Si les tailles de tableaux ne sont pas les mêmes, le maximum de données communes sera affecté.


## Exercice 2 Calcul automatique de constantes

Comme vu en cours, les templates peuvent être utilisés pour faire de la métaprogrammation, c'est-à-dire de déléguer une partie d'exécution au compilateur. Typiquement, la métaprogrammation peut être utilisée pour calculer des valeurs constantes à l'exécution du programme.

Dans cet exercice, nous allons utiliser la métaprogrammation pour calculer la valeur d'une factorielle.

RAPPEL : la valeur de la factorielle d'un nombre entier naturel  $n$  est donnée par la formule de récurrence suivante :

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n * (n - 1)! & \text{sinon} \end{cases}$$

-  Ecrire une fonction itérative permettant de calculer une factorielle

- ➡ Le terme « itératif » signifie que vous devez faire une boucle...
- 📄 Ecrire une fonction récursive permettant de calculer une factorielle
  - ➡ Le terme « récursif » signifie que la boucle précédente doit être remplacée par un sous appel de la même fonction.
- 📄 Faites en sorte qu'une constante dont la valeur est une factorielle puisse être calculée par le compilateur
  - ➡ Vous devez faire un petit peu de métaprogrammation
  - ➡ Utilisez ce mécanisme pour calculer la valeur de 12!
  - ➡ Testez avec des valeurs plus grandes et expliquez le résultat.