

TP Noté CPOA

Ecrire une classe **Pile** paramétrée par le type des éléments qu'elle contient.

On l'implantera sous la forme d'un tableau alloué dynamiquement qui sera réalloué quand la taille est insuffisante. Les allocations se feront par zone de 10 éléments. La mémoire sera libérée uniquement à la destruction de la pile.

On utilisera le projet fourni, en essayant au maximum de compiler le code d'exemple du *main()*
La sortie attendue est dans le fichier *output.txt*.

On écrira :

- le constructeur
- le constructeur de copie
- l'opérateur d'affectation
- la méthode **head** qui renvoie la tête de la pile
- la méthode **push** qui empile un élément
- la méthode **pop** qui enlève la tête de la pile
- la méthode **empty** qui renvoie vrai si la pile est vide
- la méthode **size** qui renvoie le nombre d'élément sur la pile
- l'opérateur de flux de sortie
- l'opérateur + qui concatène deux piles (ordre conservé)
- l'opérateur [] qui renvoie les éléments dans l'ordre d'insertion. On ne pourra pas modifier les éléments de la pile renvoyés par cette méthode.
- l'opérateur () qui renvoie les éléments dans l'ordre inverse ((0) renvoie la tête). On ne pourra pas modifier les éléments de la pile renvoyés par cette méthode.
- une fonction (externe à la classe) template **convert** qui convertit une *Pile<T1>* en *Pile<T2>*, pour faire par exemple *Pile<int> pi ; ... ;Pile<float> pf = convert(pi) ;*
- Définir les types **PileInt** et **PileString**

Dériver la classe **PileOper** qui travaille sur des float en implantant les fonctionnalités suivantes :

- le constructeur par copie (paramètre une ref sur un objet de la classe mère).
- la méthode **compactSomme** qui dépile tous les éléments et empile leur somme.
- l'opérateur * qui fait le produit " élément à élément"
- une fonction **scalaire** qui fait le produit scalaire de 2 *PileOper* (en utilisant les méthodes précédentes)

En utilisant le SFINAE, faire **PileOperGen** et **scalaire_gen** qui soient utilisable avec tout type numérique.

On prendra soin de mettre autant de **const** et de **&** que possible (quand cela a du sens).

On utilisera des **assert** partout ou cela semble nécessaire.

On commentera le code si jugé nécessaire.