Travaux Dirigés de C++.

Maitrise Informatique.

—*Julien Allali* - `allali@univ-mlv.fr` —

▸ **Exercice 1.** `Array` *:*
Ecrire une classe `Array` *générique se comportant comme un tableau. On écrira entre autre l'opérateur* `[]` *permettant d'accéder aux objets du tableau. Dans le cas où l'opérateur* `[]` *est appelé avec un indice supérieur à la taille actuelle du tableau, on définira deux comportements:*

- *Si le tableau est constant, alors on lève une exception de type* `OutOfBoundException`*.*
- *Si le tableau n'est pas constant, alors on retaille le tableau pour que l'indice demandé soit valide.*

▸ **Exercice 2.** *Spécialisation:*
On souhaite maintenant optimiser la classe `Array` *dans 2 cas:*

Si le type des objets mis dans le tableau est `bool` *alors on utilisera un tableau de caractères en interne, chaque caractère repésentera huit entrées du tableau (1 bit = une entrée).*

Si les objets mis dans le tableau sont des pointeurs, alors on veillera à initialiser les cases du tableau à `NULL` *lors de leurs créations.*

▸ **Exercice 3.** *Meta-programmation:*
Toujours sur la classe `Array`*, dans le cas où le tableau contient des objets de grande taille (dont la taille est supérieur à une constante fixée), on souhaite manipuler les objets à travers un tableau de pointeurs.*

Pour cela écrire une classe `Sup` *paramétrée par deux entiers contenant une variable constante static public de type* `bool` *valant* `true` *si le premier entier est supérieur au deuxième et* `false` *sinon.*

Écrire une classe `InternalArray` *paramétrée par un type* `T` *et un booléen* `pointeur`*. Cette classe utilisera en interne un tableau de* `T` *si* `pointeur` *vaut* `false` *et un tableau de pointeurs sur* `T` *si* `pointeur` *vaut* `true`*.*

Pour finir modifier la classe `Array` *pour qu'elle utilise la classe* `InternalArray`*.*

▸ **Exercice 4.** *Petit détail:*
Comment faire pour que la taille de l'objet puisse être spécifiée par l'utilisateur. Celle-ci vaudra toujours sizeof(T) *par défault.*

▸ **Exercice 5.** *Pour aller plus loin:*

Reprendre la classe `InternalArray` *dans le cas de l'implémentation par pointeurs (objets de grande taille). Implémenter un mécanisme type* compteur de références *permettant de faire du* copy-on-write.

Voici un exemple de copy-on-write*: on créer un* `Array` *A de 200 matrices de taille* 100*100. *On crée maintenant un nouvel objet* `Array` *B par recopie de A. À ce moment là, les 200 matrices ne sont pas vraiment copiées, A et B partagent les même matrices. Si l'on fait un accès à travers B de la matrice à l'indice* 10*, celle-ci n'est pas copiée. Si l'on fait un accès en écriture sur la matrice d'indice* 20 *de A, alors A se fait une copie de la matrice puisque le compteur de références lui indique qu'un autre* `Array` *partage cette matrice.*

▸ **Exercice 6.** *D'accord mais...*

Il arrive souvent que l'on utilise l'opérateur `[]` *sur un* `Array` *non constant alors qu'on ne compte pas modifier l'objet. Comment faire pour détecter les cas d'utilisation de l'opérateur* `[]` *pour faire une modification indépendemment du contexte* const*?*

Implémenter votre solution...