Exercices de C++ - jour 2

Gestion des opérateurs :

Exercice 1 :
Écrivez un foncteur qui peut être utilisé avec *std::sort* pour trier un vecteur de chaînes de caractères en fonction de leur longueur, de la plus courte à la plus longue.

Exercice 2 :
Écrivez un foncteur qui peut être utilisé avec *std::remove_if* pour filtrer un vecteur de nombres entiers. Ce foncteur devrait permettre de supprimer tous les nombres impairs du vecteur.

Exercice 3 :
Écrivez un foncteur qui agit comme un « incrémenteur ». Il doit pouvoir être utilisé pour accumuler des valeurs numériques entières en maintenant un total. Le foncteur devra avoir une méthode pour ajouter une valeur et une méthode pour récupérer le total accumulé.

Exercice 4 :
Créez une classe « ProxyArray » qui agit comme un proxy autour d'un tableau dynamique d'entiers. Le proxy devra gérer la libération de la mémoire lorsque l'objet est détruit, ainsi que le décompte des références.
Ainsi il doit inclure :

- Un pointeur pour accéder aux éléments du tableau.
- Une méthode pour obtenir l'adresse du tableau.
- Des opérateurs de comparaison (égal et différent) pour comparer deux proxies.
- Une méthode getSize() pour obtenir la taille du tableau.
- L'utilisation de l'opérateur [] pour accéder directement aux éléments du tableau.
- La surcharge de l'opérateur d'insertion pour afficher le contenu du tableau.
- La possibilité d'insérer de nouvelles valeurs dans le tableau avec l'opérateur >>.

Enfin essayez de passer le contenu d'une instance de ProxyArray au foncteur de l'exercice précédent pour voir s'il le traite correctement.

Conversion et RTTI :

Exercice 5 :
Créez une classe de base « véhicule » et deux classes dérivées « voiture » et « vélo ». Ces classes auront une méthode *displayType()* qui affichera le type du véhicule. Instanciez ensuite un objet voiture et vélo puis convertissez ces objets en « véhicule » et appelez leur méthode *displayType()*.

Exercice 6 :
Créez une classe de base Animal et deux classes dérivées Dog et Cat. Chaque classe a une méthode *makeNoise()* pour représenter le bruit que fait l'animal.
Créez ensuite un objet de type Dog et un objet de type Cat et convertissez-les en objets de type Animal. Testez la méthode *makeNoise()* puis convertissez ces objets Animal en objets Dog et Cat. Vérifiez si la conversion est réussie puis appeler la méthode *makeNoise()*.

Exercice 7 :
Vous développez une application de traitement d'images. Créez une structure Pixel qui représente un pixel avec des composantes rouge, verte et bleue de type entier.
Écrivez ensuite une fonction *changeColor()* qui prend un pointeur vers un tableau de pixels et change la couleur de chaque pixel en inversant les composantes rouge et bleue. Pour cela, effectuez un reinterpret_cast pour convertir le pointeur vers le tableau de pixels en pointeur vers un tableau de bytes.
Testez la fonction et vérifiez si les couleurs ont été correctement inversées.

P.S. pour inverser les couleurs rouge et bleu, je vous conseille de regarder **std::swap**.