TP de C++ n° 11

Exercice 1 : Définition de l'opérateur `()`

On va définir l'"interface" (i.e. classe avec seulement des méthodes virtuelles pures) ci-dessous.

```
class ValeursAdmises {//"interface"
 public :
  virtual  bool operator()(char val) = 0;
};
```

Dans cette interface, on indique que l'opérateur "`()`" est défini comme prennant un argument `val` de type `char`. En gros, cette interface représente une fonction qui va vérifier que la valeur `val` répond à certains critères.

On va écrire deux sous-classes concrètes de `ValeursAdmises` :

- `ValeursAdmisesIntervalle` qui va contenir une valeur min et une valeur max et dont la "fonction" va vérifier que la valeur donnée en argument est dans cet intervalle.
- `ValeursAdmisesTableauValeurs` qui va contenirun tableau de valeurs et dont la "fonction" va vérifier que la valeur donnée en argument est dans ce tableau.

Exemple d'utilisation :

```
ValeursAdmisesIntervalle inter('a', 'd');
 if(inter('e'))
   cout<< " la valeur 'e' est ok"<< endl;
 else
    cout<< " la valeur 'e' n'est pas ok"<< endl;
 if(inter('c'))
   cout<< " la valeur 'c' est ok"<< endl;
 else
    cout<< " la valeur 'c' n'est pas ok"<< endl;

 char tab[] = {'b', 'o', 'n', 'j', 'u', 'r'};
 ValeursAdmisesTableau tableau(tab, 6);
  if(tableau('j'))
   cout<< " la valeur 'j' est ok"<< endl;
 else
    cout<< " la valeur 'j' n'est pas ok"<< endl;
 if(tableau('c'))
   cout<< " la valeur 'c' est ok"<< endl;
 else
    cout<< " la valeur 'c' n'est pas ok"<< endl;
```

Exercice 2 :

- Reprenez l'exercice précédent en remplaçant `char` par un type `T` générique.
- Écrivez une fonction générique `filtre` qui va prendre en argument une liste `l` d'éléments de type `T` et un objet de type `ValeursAdmises<T>` et retournera une liste ne contenant que les éléments `l` qui sont admis.

On pourra tester avec des types genre `int`, `char`, mais aussi les fractions en (re)définissant les opérateurs adéquats.

Exercice 3 : Pointeurs intelligents

Dans cet exercice nous allons faire notre propre classe de pointeur intelligent, similaire à `std::unique_ptr`. Vous trouverez ici un exemple d'utilisation, puis vous suivrez les différentes étapes de l'implémentation.

```
#include <iostream>
#include <../src/Mon_ptr_u.cpp>
using namespace std;

class A{
public :
    int v;
    A(int x):v(x){}
};

template <class T> void f(Mon_ptr_u<T> x) {}
template <class T> void g(Mon_ptr_u<T> &x) {}

int main()
{
    // Illustration question 1
    A a1(1);
    Mon_ptr_u<A> p1(&a1);
    // Illustration question 2
    // Mon_ptr_u<A> p1bis(p1); // ne doit pas marcher
    // f(p1); // ne doit pas marcher
    g(p1);  // est OK
    // Illustration question 3
    cout << "---------" << endl;
    cout << (p1.release())->v << endl;
    //cout << (p1.release())->v << endl;  // ne marche plus
    cout << p1.release() << endl; // affiche le 0 correspondant à nullptr
    // Illustration question 4
    cout << "---------" << endl;
    A a2(2);
    Mon_ptr_u<A> p2(&a2);
    p1=p2;
    // cout << (p2.release())->v << endl; ne doit pas marcher
    cout << p2.release() << endl; // affiche le 0 correspondant à nullptr
    cout << (p1.release())->v << endl; // affiche 2
    cout << p1.release() << endl; // affiche le 0 correspondant à nullptr
    Mon_ptr_u<A> p3(new A(3));
    p3=p3;
    cout << p3.release()->v << endl; // affiche 3
    cout << p3.release() << endl; // affiche le 0 correspondant à nullptr
    p3=p3;
    cout << p3.release() << endl; // affiche le 0 correspondant à nullptr
    p3=p3;
    // Illustration question 5
    cout << "---------" << endl;
    Mon_ptr_u<A> p4(new A(4)), p5 (new A(5));
    p4.echange(p5);
    cout << p4.release()->v << endl; // affiche 5
    cout << p5.release()->v << endl; // affiche 4
    // Illustration question 6
```

```
    cout << "---------" << endl;
    Mon_ptr_u<A> p6(new A(6));
    if (p6) cout << "p6 pointe vers la valeur " << p6.release()->v << endl;
        else cout << "p6 pointe vers nullptr" << endl;
    if (p6) cout << "p6 pointe vers la valeur " << p6.release()->v << endl;
        else cout << "p6 pointe vers nullptr" << endl;
    // Illustration question 7
    cout << "---------" << endl;
    Mon_ptr_u<A> p7(new A(7));
    cout << "p7 contient " << p7->v << endl;
    cout << "p7 contient " << (*p7).v << endl;
    return 0;
}
```

1. déclarez une classe générique `Mon_ptr_u` qui encapsule une pointeur vers un objet de classe `T`. Quelle est la visibilité pour ce champs ? Ecrivez le destructeur
2. Puisqu'on cherche à garantir qu'il n'y aura pas d'autres pointeurs intelligents vers le même objet, il vous faut en particulier "désactiver" le constructeur de copie. Comment procédez vous ?
3. Ecrivez une méthode `release()` qui retourne le pointeur encapsulé, et qui annule le contenu du smart pointeur en lui affectant la valeur `nullptr`
4. L'opérateur d'affectation de notre classe peut être utilisé mais dans une sémantique différente, celle dite du "Move-assignement". Ainsi le sens de l'affectation `x=y` sera de transférer à x le rôle de représenter le pointeur encapsulé par y.
   Le résultat sera que y ne représentera plus personne (c'est à dire `nullptr`), et que l'ancienne référence pointée par x sera libérée. (Assurez vous que votre code se comporte bien au cas où l'on écrive $x = x$, ou dans le cas où x était un pointeur intelligent vers `nullptr`)
5. Ecrivez une méthode `echange(Mon_ptr_unique )` qui permute les éléments pointés.
6. Pour un pointeur `p` classique, on peut par exemple écrire `while(p){...}` ou `if (p) {...}` C'est à dire qu'on peut convertir un pointeur vers une valeur de vérité. Redéfinissez l'opérateur `operator bool() const` pour que l'on puisse faire de même avec notre pointeur intelligent.
7. Redéfinissez les opérateur `*` et `->` pour que si m_p est un élément de votre classe, alors `*m_p` et `m_p->xxx` aient le sens attendu. Il s'agit de réécrire `operator*() const` et de `operator->() const`
8. (optionnel) Si on veut réellement assurer l'unicité, on ne devrait pas pouvoir construire deux fois un élément de votre classe à partir du même pointeur. Ajoutez la gestion d'une liste statique des adresses déjà encapsulées. Si un utilisateur construit deux smarts pointeurs identiques, levez une exception.