TP 3 - Premières classes

Histoire d'ensembles

1 Ensemble d'entiers

Définir une classe `Ensemble` pour manipuler des ensembles d' "elements" (`#define elements int` dans un premier temps). On utilisera l'allocation dynamique pour manipuler des ensembles de taille quelconque.

1) Donnez la définition de la classe dans un premier temps sans son implémentation. On considère pour le moment que la classe Ensemble ne contient que les opérations `appartient`, `card` et `inserer`.

2) L'utilisation de l'allocation dynamique implique d'initialiser les données membres de la classe lors de sa création. Pour cela, il vous faut modifier le constructeur par défaut, ainsi que le destructeur.

3) Ajouter une méthode `affiche()` pour la classe.

4) Surchargez la méthode `affiche()` de manière à ce qu'elle puisse prendre comme paramètre le rang d'un élément et afficher cet élément.

5) Ajouter un programme principal utilisant la classe `Ensemble`. Créez deux éléments. Donnez leur une valeur. Ajouter les à l'ensemble. Affichez.

6) Ajouter une méthode `void retirer(const ensemble e)`.

```
#ifndef ENSEMBLE_INT_H
#define ENSEMBLE_INT_H
#include<iostream>
using namespace std;

#define element int

class Ensemble{
        element * tab;
        int taille;
  public :
        int card() const;
        bool appartient (const element e) const ;
        void inserer (const element e) ;
        void retirer (const element e) ;
        void affiche () const;
        void affiche (const int iRang) const;
        Ensemble();
        ~Ensemble();
};
#endif
```

```
#include "ensembleInt.h"

/*************************************************************/
/* Redefinition du constructeur par defaut afin de lui permettre
d'initialiser les donnees membres */
Ensemble::Ensemble(){
   taille=0;
   tab=NULL;
}

/*************************************************************/
/* Redefinition du destructeur par defaut afin de lui permettre
de liberer les donnees allouees dynamiquement */
Ensemble::~Ensemble(){
   taille=0;
   delete [] tab;
   tab=NULL;
}

/*************************************************************/
/* L'operation est specifiee comme const car elle ne modifie pas
l'ensemble */
int Ensemble::card() const{
```

```
  return taille;
}

/*************************************************************/
/* Le parametre e1 est specifie comme const car il n'est pas modifie par l'operation */
bool Ensemble::appartient(const element el) const{
        if (tab!=NULL)
                for (int i=0; i< taille; i++)
                        if (tab[i]==el)
                                return true;
         return (false);
}

/*************************************************************/
/* L'operation n'est pas specifiee comme const car elle modifie l'ensemble */
void Ensemble::inserer (const element e){
    if (!appartient(e)){
                //tab=(element*)realloc(tab,(++taille)*sizeof(element));
                // le new ne permet pas de faire de la reallocation memoire
                // il faut donc allouer un nouveau bloc pour la nouvelle taille
                // et recopier la totalite du precedent tableau dans le nouveau
                // et enfin liberer l'ancien
                // ou alors utiliser le template vector
      element * tab1 = new element [++taille];  // allocation tab temporaire
          // recopie des elements de tab dans tab1
          for (int i=0; i<taille-1;i++){
                tab1[i] = tab[i];
          }
          //affectation du nouvel element
      tab1[taille-1]=e;
          // liberation de la memoire reservee par l ancien tableau
          delete [] tab;
          // affectation du nouveau tableau
          tab=tab1;
    }
}


/*************************************************************/
/* L'operation n'est pas specifiee comme const car elle modifie l'ensemble */
void Ensemble::retirer (const element e){

    if (appartient(e)){
                int j=0;
                int iNewTaille = taille-1; //determination de la taille du nouveau tableau

      element * tab1 = new element [iNewTaille];        // allocation tab temporaire

          // recopie des elements de tab dans tab1
          for (int i=0; i<taille;i++){
                if (tab[i]!=e){
                        tab1[j] = tab[i];
                        j++;
                }
          }

          // on positionne la nouvelle taille dans l objet courant
          taille=iNewTaille;

          // liberation de la memoire reservee par l ancien tableau
          delete [] tab;
          // affectation du nouveau tableau
          tab=tab1;
    }
}
/*************************************************************/
void Ensemble::affiche () const{
        for (int i=0; i<taille; i++)
                cout << tab[i] << ' ';
        cout << endl;
}

/*************************************************************/
void Ensemble::affiche (const int iRang) const{
        cout << tab[iRang] << endl;
}
```

```
#include "ensembleInt.h"

int main(){
        cout << "*******************************************"<<endl;
```

```
        Ensemble ens1;
        element e1, e2, e3;

        e1 = 3;
        e2 = 7;
        e3 = 10;
        cout << endl << "cardinalite de l ensemble : " << ens1.card() << endl;
        cout << "Ajout des elements : " << e1 <<  " " << e2 << " "  << e3 << endl;
        ens1.inserer(e1);
        ens1.inserer(e2);
        ens1.inserer(e3);

        cout << "cardinalite de l ensemble : " << ens1.card() << endl;

        cout << "Ensemble ens1 = ";
        ens1.affiche();

        cout << endl << "Retrait de l elements : " << e2 << endl;
        ens1.retirer(e2);

        cout << "cardinalite de l ensemble : " << ens1.card() << endl;
        cout << "Ensemble ens1 = ";
        ens1.affiche();

        cout << endl;
        cout << "*********************************************"<<endl;
        return EXIT_SUCCESS;
}
```

2 Ensemble complexe

Modifier l'exercice précédent de manière à définir et manipuler un ensemble de couleurs. Partez de la classe `Couleur.h` donnée ci-dessous :

```
#ifndef COULEUR_H
#define COULEUR_H
#include<iostream>
using namespace std;

class Couleur{
        string sCouleur;
  public :
        string getCouleur () const;
        void setCouleur (const string & s);
};
#endif
```

Que devez-vous modifier ou ajouter comme méthodes à la classe ci-dessus afin de permettre son utilisation avec la classe `Ensemble` ? Faites-le et faites faire des affichage à chaque méthode afin de constater leurs appels..

****Remarque :

- il faut passer la classe sous la forme normale de Coplien a minima !
- pour la méthode *appartient()*, il faut également surcharger l'opérateur d'égalité : *operator==*
- pour la méthode *retirer()*, il faut également surcharger l'opérateur de différence : *operator !=*

```
#ifndef COULEUR_H
#define COULEUR_H
#include<iostream>
using namespace std;

class Couleur{
        string sCouleur;
  public :
        string getCouleur () const;
        void setCouleur (const string & s);
// modification de la classe
        Couleur ();
        Couleur (const Couleur &);
        ~Couleur ();
        Couleur &operator=(const Couleur);       // necessaire pour faire c1=c2
        Couleur &operator=(const string);        // necessaire pour faire c1="bleu"
        friend ostream & operator<<(ostream &, Couleur &);
        friend bool operator== (Couleur const c1, Couleur const c2);
        friend bool operator!= (Couleur const c1, Couleur const c2);
};
#endif
```

```
#include "Couleur.h"
////////////////////////////////////////////////////////
string Couleur::getCouleur () const{
        return sCouleur;
}
////////////////////////////////////////////////////////
void Couleur::setCouleur (const string & s){
        sCouleur = s;
}

/*************** Classe modifiee *********************/
////////////////////////////////////////////////////////
Couleur::Couleur (){
#ifdef DEBUG
        cout << "DEBUG - Appel constructeur de Couleur" << endl;
#endif
}
////////////////////////////////////////////////////////
Couleur::Couleur (const Couleur & c){
#ifdef DEBUG
        cout << "DEBUG - Appel constructeur par recopie de Couleur"     << endl;
#endif

        sCouleur=c.sCouleur;
}
////////////////////////////////////////////////////////
Couleur::~Couleur (){
#ifdef DEBUG
        cout << "DEBUG - Appel destructeur de Couleur"  << endl;
#endif

}
////////////////////////////////////////////////////////
Couleur & Couleur::operator=(const Couleur c ){
        #ifdef DEBUG
        cout << "DEBUG - Appel operateur affectation de Couleur"        << endl;
#endif
        sCouleur=c.sCouleur;
        return *this;
}
////////////////////////////////////////////////////////
Couleur & Couleur::operator=(const string s ){
        #ifdef DEBUG
        cout << "DEBUG - Appel operateur d affectation directe de Couleur"      << endl;
#endif
        sCouleur=s;
        return *this;
}
////////////////////////////////////////////////////////
ostream & operator<<(ostream & out, Couleur & c){
        out << c.sCouleur << endl;
        return out;
}
////////////////////////////////////////////////////////
bool operator== (Couleur const c1, Couleur const c2){
        return (c1.sCouleur == c2.sCouleur);
}
////////////////////////////////////////////////////////
bool operator!= (Couleur const c1, Couleur const c2){
        return (c1.sCouleur != c2.sCouleur);
}
```

```
#ifndef ENSEMBLE_COULEUR_H
#define ENSEMBLE_COULEUR_H
#include<iostream>
#include "Couleur.h"
using namespace std;

#define element Couleur

class Ensemble{
        element * tab;
        int taille;
  public :
        int card() const;
        bool appartient (const element e) const ;
        void inserer (const element e) ;
        void retirer (const element e) ;
        void affiche () const;
        void affiche (const int iRang) const;
        Ensemble();
```

```
        ~Ensemble();
};
#endif

#include "ensembleCouleur.h"

/*************************************************************/
/* Redefinition du constructeur par defaut afin de lui permettre
d'initialiser les donnees membres */
Ensemble::Ensemble(){
  taille=0;
  tab=NULL;
}

/*************************************************************/
/* Redefinition du destructeur par defaut afin de lui permettre
de liberer les donnees allouees dynamiquement */
Ensemble::~Ensemble(){
  taille=0;
  delete [] tab;
  tab=NULL;
}

/*************************************************************/
/* L'operation est specifiee comme const car elle ne modifie pas
l'ensemble */
int Ensemble::card() const{
  return taille;
}

/*************************************************************/
/* Le parametre e1 est specifie comme const car il n'est pas modifie par l'operation */
bool Ensemble::appartient(const element el) const{
        if (tab!=NULL)
                for (int i=0; i< taille; i++)
                        if (tab[i]==el)
                                return true;
         return (false);
}

/*************************************************************/
/* L'operation n'est pas specifiee comme const car elle modifie l'ensemble */
void Ensemble::inserer (const element e){
    if (!appartient(e)){
                //tab=(element*)realloc(tab,(++taille)*sizeof(element));
                // le new ne permet pas de faire de la reallocation memoire
                // il faut donc allouer un nouveau bloc pour la nouvelle taille
                // et recopier la totalite du precedent tableau dans le nouveau
                // et enfin liberer l'ancien
                // ou alors utiliser le template vector
      element * tab1 = new element [++taille];  // allocation tab temporaire
          // recopie des elements de tab dans tab1
          for (int i=0; i<taille-1;i++){
                tab1[i] = tab[i];
          }
          //affectation du nouvel element
      tab1[taille-1]=e;
          // liberation de la memoire reservee par l ancien tableau
          delete [] tab;
          // affectation du nouveau tableau
          tab=tab1;
    }
}


/*************************************************************/
/* L'operation n'est pas specifiee comme const car elle modifie l'ensemble */
void Ensemble::retirer (const element e){

    if (appartient(e)){
                int j=0;
                int iNewTaille = taille-1; //determination de la taille du nouveau tableau

      element * tab1 = new element [iNewTaille];        // allocation tab temporaire

          // recopie des elements de tab dans tab1
          for (int i=0; i<taille;i++){
                if (tab[i]!=e){
                        tab1[j] = tab[i];
                        j++;
                }
```

```
            }

            // on positionne la nouvelle taille dans l objet courant
            taille=iNewTaille;

            // liberation de la memoire reservee par l ancien tableau
            delete [] tab;
            // affectation du nouveau tableau
            tab=tab1;
    }
}
/*************************************************************/
void Ensemble::affiche () const{
        for (int i=0; i<taille; i++)
                cout << tab[i] << ' ';
        cout << endl;
}

/*************************************************************/
void Ensemble::affiche (const int iRang) const{
        cout << tab[iRang] << endl;
}
```

```
#include "ensembleCouleur.h"

int main(){
        cout << "*********************************************"<<endl;
        Ensemble ens1;
        element e1, e2, e3;

        e1 = "bleu";
        e2 = "jaune";
        e3 = "vert";
        cout << endl << "cardinalite de l ensemble : " << ens1.card() << endl;
        cout << "Ajout des elements : " << e1 <<  " " << e2 << " "  << e3 << endl;
        ens1.inserer(e1);
        ens1.inserer(e2);
        ens1.inserer(e3);

        cout << "cardinalite de l ensemble : " << ens1.card() << endl;

        cout << "Ensemble ens1 = ";
        ens1.affiche();

        cout << endl << "Retrait de l elements : " << e2 << endl;
        ens1.retirer(e2);

        cout << "cardinalite de l ensemble : " << ens1.card() << endl;
        cout << "Ensemble ens1 = ";
        ens1.affiche();

        cout << endl;
        cout << "*********************************************"<<endl;
        return EXIT_SUCCESS;
}
```