Compte Rendu TP4 CPOO

Théo CHAPON, Hassan EL OMARI ALAOUI

INSA de Rennes
4INFO, groupe 1.1

16 octobre 2014

TP4 : Création et utilisation d'un template

1 Etat du TP

Le projet est terminé, fonctionel et testé.

2 Objectif

Ce TP avait pour but d'apprendre à utiliser et créer une classe template. Dans notre cas, il s'agit d'une classe template "Ensemble" qui utilise la classe template "List".

3 Réalisation

Nous devions réaliser un template permettant de représenter un ensemble d'objets non trié et sans doublon. Pour gérer les ensembles, nous devions surchager les opérateurs arithmétiques de base :

- '+' pour l'union de deux ensembles
- '*' pour l'intersection de deux ensembles
- '-' pour la soutraction de deux ensembles
- '/' pour la différence de deux ensembles
- '«' pour gérer l'affichage des ensembles
- '»' pour gérer la création par fichier d'un ensemble

Nous devions aussi produire un constructeur par recopie. La totalité des fonctions du template "Ensemble" ont utilisé des méthodes provenant de la classe "List".

Une seul petite contrainte a été rencontrée dans ce TP, c'est l'utilisation d'une énumération de la classe "List" protéger par protected. Cela nécessite de faire de l'héritage mais ça complique le code de la classe "Ensemble".

Listing 1 – Définition de la class template Ensemble

```
#pragma once
#include "stdafx.h"
#include "list.h"

using namespace std;
template <class T>
// == operator required
class Ensemble {
private:
  List<T> _ensemble;
  ostream& _print(ostream& os) {
    return os << _ensemble;
  }
public:
  Ensemble<T>():_ensemble() {}
  Ensemble<T>(const Ensemble<T>& e):_ensemble(e._ensemble) {}
  ~Ensemble<T>(){}

  bool exists(T value) {
    for (ListIterator<T> it = _ensemble.beg(); !it.finished(); ++it){
      if (it.get() == value)
      {
        return true;
      }
    }
    return false;
  }

  friend Ensemble<T>& operator+=(Ensemble<T>& e1, Ensemble<T>& e2) {
    for (ListIterator<T> it = e2._ensemble.beg(); !it.finished(); ++it)
    {
      if (!e1.exists(it.get()))
      {
        e1._ensemble.addElement(it.get()/*, List<T>::LP_last*/);
      }
    }
    return e1;
  }
```

```
39    friend Ensemble<T> operator+(const Ensemble<T>& e1, Ensemble<T>& e2) {
40      Ensemble<T> e3(e1);
41      return e3 += e2;
42    }
43
44    friend Ensemble<T>& operator-=(Ensemble<T>& e1, Ensemble<T>& e2){
45      for (ListIterator<T> it = e2._ensemble.beg(); !it.finished(); ++it)
46      {
47        e1._ensemble.delElement(it.get());
48      }
49      return e1;
50    }
51    friend Ensemble<T> operator-(const Ensemble<T>& e1, Ensemble<T>& e2){
52      Ensemble<T> e3(e1);
53      return e3 -= e2;
54    }
55
56    friend Ensemble<T>& operator*=(Ensemble<T>& e1, Ensemble<T>& e2) {
57      Ensemble e3(e1);
58      for (ListIterator<T> it = e3._ensemble.beg(); !it.finished(); ++it)
59      {
60        if (!e2.exists(it.get()))
61        {
62          e1._ensemble.delElement(it.get());
63        }
64      }
65      return e1;
66    }
67    friend Ensemble<T> operator*(const Ensemble<T>& e1, Ensemble<T>& e2) {
68      Ensemble<T> e3(e1);
69      return e3 *= e2;
70    }
71
72    friend Ensemble<T>& operator/=(Ensemble<T>& e1, Ensemble<T>& e2){
73      return e1 = (e1 + e2) - (e1 * e2);
74    }
75    friend Ensemble<T> operator/(const Ensemble<T>& e1, Ensemble<T>& e2){
76      Ensemble<T> e3(e1);
77      return e3 /= e2;
78    }
79    template<typename T> friend istream& operator>>(istream& is, Ensemble<T>& e);
80    template<typename T> friend ostream& operator<<(ostream& os, Ensemble<T>& e);
81
82  };
83  template <class T>
84  istream& operator>>(istream& is, Ensemble<T>& e){
85    return is >> e._ensemble;
86  }
87  template <class T>
```

```
88 ostream& operator<<(ostream& os, Ensemble<T>& e) {
89   return e._print(os);
90 }
```