Université Paris Diderot **Langages à objets avancés**
M1 **Année 2016-2017**

TP n° 11: itérateurs-template

Exceptions en C++

Dans se sujet nous aurons besoin d'exceptions. La syntaxe de `throw` et de `try ... catch` est la même qu'en Java. La seule différence est qu'on peut lancer tout type de données à partir du moment où elle peut être copiée. Il est cependant recommandé de créer une classe spéciale à cette effet.

Sujet

On veut représenter des suites d'objets d'un type donné. On va donc définir l'"interface" `Suite` comme suit:

```
template <class T> class Suite
{
public:
  virtual T &operator[] (const unsigned int)=0; //doit permettre de modifier la valeur
  virtual int taille() = 0; //nombre d'éléments "réels"
  virtual void affiche()=0;//affiche les éléments
  virtual void ajoute(T)=0;// ajoute à la fin
};
```

L' "interface" parcours, est une sorte d'itérateur (inspiré en partie du `iterator` de la STL).

```
template <class T> class Parcours
{
public:
  virtual bool estFini()= 0;

  //avance d'une case, opérateur préfixé, renvoie une lvalue
  virtual Parcours<T> &operator++()=0;

  //avance d'une case, opérateur postfixé, renvoie une rvalue
   virtual  Parcours<T> & operator++(int)=0;

  //déréférencement
  virtual T& operator*()=0; //déréférencement

  //recommence le parcours à partir du début
  virtual void retourneDebut()=0;

  // doit permettre d'écrire *(it+2)= elem
  virtual Parcours<T> &operator+(int n)=0;

};
```

Exercice 1 Dérivez la classe `Suite` en une classe `SuiteTab` en utilisant un tableau comme structure de données. Il y aura un constructeur qui prendra en argument la

taille du tableau et `ajoute()` enverra une exception quand le tableau sera plein. Il sera peut-être judicieux que d'autres méthodes envoient des exceptions.

Dérivez la classe `Parcours` en une classe `ParcoursTab` qui représentera un parcours sur une `SuiteTab`.

Il est à noter que certaines méthodes de `ParcoursTab` engendreront des fuites de mémoire. Lesquelles?

On ajoutera ensuite dans `SuiteTab` une méthode de prototype `ParcoursTab<T> parcours();` qui retournera un parcours sur la suite courante. On ne peut pas déclarer cette méthode dans `Suite` car on ne peut pas faire retourner un type abstrait (ici `Suite<T>`) par une méthode virtuelle. ce serait possible si on ajoutait une référence, mais pour ne pas ajouter encore des fuites de mémoire nous ne le feront pas.

Pour gérer le problème des dépendances croisées dans des templates

Tout le code d'un template sera dans un fichier `hpp`.

Tout d'abord la syntaxe pour donner le code d'une méthode en dehors de la définition de classe est:

```
template <class T>  SuiteTableau<T>::SuiteTableau(){...}
template <class T>  Parcours<T> & SuiteTableau<T>::parcours(){....}
```

Ensuite on fait comme d'habitude: dans suiteTab on déclare `template <class T> class ParcoursTab;`, on donne la définition de la classe et avant le code des méthodes on fait `#include "ParcoursTab.hpp"`. Dans `ParcoursTab` on fera le `#include "SuiteTab.hpp"` dès le début.

Il est aussi possible de faire un fichier cpp avec le code des méthodes, et de l'inclure dans le `hpp` à la fin du fichier.

Exercice 2 Dérivez la classe `Suite` par une interface `SuiteTriee` qui représente des suites triées. Dans cette interface, la méthode `ajoute`, insère l'élément de telle manière que la suite reste triée.

Dérivez cette classe avec une implémentation à l'aide de tableau.