Représentation des suites

Travaux dirigés et pratiques - Programmation par objets 2 (UE ULIN606)

Lors de ces travaux dirigés, nous allons retravailler sur l'héritage et faire un exercice préparatoire au cours portant sur la généricité paramétrique (*templates* en dialecte C++).

1 La classe paramétrée `vector`

C++ dispose, comme Java depuis la version 1.5, d'une librairie de structures de données paramétrées que nous explorerons plus en détails dans un prochain cours. Pour s'y préparer, nous commençons par utiliser la classe paramétrée `vector`. Le paramètre dont nous parlons ici est le type des objets qui seront placés dans le vecteur.
Nous présentons tout d'abord une petite partie de l'interface de `vector` qui sera suffisante pour les exercices suivants.

```
template<class T>
class vector
{
private:
...
public:
  vector(); //crée un vecteur vide
  ...
  virtual bool empty()const; //retourne vrai si et seulement si le vecteur est vide
  virtual int size()const; //retourne le nombre d'éléments stockés
  virtual void push_back(const T& e); //ajout de l'élément e en fin du vecteur
  virtual void pop_back(); // enlève le dernier élément
  virtual T& back(); // accès par référence au dernier élément
                     // (qui doit exister)
  virtual T& operator[](int i); //accès par référence au ième élément
                                //(qui doit exister)
};
```

Voici également un exemple d'utilisation de cette classe paramétrée.

```
main()
{
  vector<int> v; // déclaration et création d'un vecteur d'entiers
  v.push_back(4); // ajout de 4 à la fin de v
  v.push_back(5);
  v.push_back(6);
  int d=v.back(); // récupération du dernier élément de v
  cout << "element qui va etre enleve = " << d << endl;
  v.pop_back();
  // affichage du contenu de v et appel de l'opérateur []
  for (int i=0; i<v.size(); i++) cout << v[i] <<" "; cout << endl;
}
```

L'exécution du programme affiche les deux lignes suivantes :

```
element enleve = 6
4 5
```

2 Les suites

On souhaite représenter des suites réelles, et plus précisément deux cas particuliers : les suites constantes à partir d'un certain rang et les suites arithmétiques.

On rappelle que ...

Une *suite réelle* est une application d'une partie de $\mathcal{N}$ dans $\mathcal{R}$. On considère ici les suites définies sur $\mathcal{N}$. Une suite ne peut pas toujours se décrire de manière finie par une relation de récurrence, une fonction, une raison et un premier terme (comme les suites arithmétiques), etc. On ne peut donc pas prévoir a priori un mode de stockage valable pour toutes les suites.

Une suite est *constante à partir d'un certain rang* s'il existe un entier naturel n et un réel a tels que pour tout entier naturel $n' \geq n$, $u(n') = a$.

Une suite est *arithmétique* si la différence de deux termes quelconques de la suite est une constante r que l'on appelle la raison de la suite. Une suite arithmétique vérifie les deux propriétés suivantes :
- $\forall n, u_n = u_0 + n.r$
- la somme des n premiers termes vaut $n.u_0 + n.(n-1).r/2$

3 Spécifications

Soit une suite `u` ; on doit pouvoir lui appliquer les opérations suivantes :
– calcul du n$^{\text{ième}}$ élément de la suite : `u(n)` (surcharge de l'opérateur `()`)
– calcul de la somme des n premiers éléments : `u.s(n)`
– si `u` est une suite arithmétique, accès à la raison : `u.raison()`
– si `u` est une suite constante à partir d'un certain rang : `u.rang()`
(par exemple pour la suite (1 9 3 1 2 9 9 9 9 9 ...), le rang sera 5).

À partir de ces descriptions d'opérations, proposez des classes permettant de représenter les deux types de suites présentés ; faites vos choix (de classes, d'attributs, de méthodes) de façon à ce que l'ajout d'autres types de suites puisse se faire facilement. Il est fortement conseillé d'utiliser la classe `vector` pour les suites constantes à partir d'un certain rang.
Pensez à la surcharge des opérateurs de lecture et d'écriture, réfléchissez aux constructeurs par copie, opérateur d'affectation, destructeurs.

4 Implémentation

Faites l'implémentation des classes que vous avez spécifiées.