TP 10 : Une classe rationnel

notions : Constructeurs, accesseurs, mutateurs, méthodes et fonctions amies, opérateurs, attributs et membres de classe

---

Vous connaissez probablement tous la définition de l'ensemble e $\mathbb{Q}$ des rationnels. Un nombre rationnel est un nombre qui peut être défini par un ratio n/d, où n est un entier, nommé numérateur, et d est un entier non nul, nommé dénominateur. Nous allons dans ce TP construire et tester une classe C++ représentant les nombres rationnels.

1 La classe `Rational` : on commence doucement

Dans un fichier d'en-tête *rational.h* et un fichier source *rational.cpp*, écrire une classe `Rational` possédant :

- Deux attributs entiers `num` et `denom`.
- Quatre constructeurs :
  - un constructeur sans paramètre (dit *"par défaut"* en C++) qui initialise le rationnel à la valeur $0/1$;
  - un constructeur avec un paramètre entier n qui initialise le rationnel à la valeur entière $n/1$;
  - un constructeur à deux paramètres *num* et *denom* qui initialise la rationnel à la valeur $num/denom$.
  - un constructeur de copie, de prototype `Rational(const Rational & other)`, qui initialise `*this` à partir du rationnel `other`.
- Un destructeur, qui se contente d'afficher `"destructeur de Rational"` dans le Terminal.
- Deux accesseurs ... `getNum()` ... et ... `getDenom()` ... retournant respectivement le numérateur et le dénonimateur.
- Deux modifieurs ... `setNum(...)` et ... `setDenom(...)` qui permettent respectivement de modifier le numérateur et le dénonimateur.
- Un accesseur `double toDouble()` ... qui retourne la valeur réelle approchée du nombre rationnel [1] .
- Un accesseur `afficher()` ... qui affiche une chaine de caractère représentant l'état du rationnel dans le Terminal, par exemple `"(<numerateur> / <denominateur>)"`.

Dans un fichier *test1.cpp*, écrire un programme principal `main()` qui :

- Crée les trois objets rationnels $r1 = 0/1$ $r2 = 3/1$ et $r3 = 5/6$.
- Affiche leurs représentations textuelles dans le Terminal
- Affiche le numérateur de $r1$
- Affiche la valeur réelle approché de $r3$
- Modifie le numérateur de $r3$ à la valeur 9 et affiche
- Modifie le dénominateur de $r2$ à la valeur 0 et affiche

Compiler et tester. Pour compiler, vous pouvez récupérer un fichier Makefile d'un TP précédent et l'adapter.

Grâce à la notion de "valeur par défaut des paramètres des fonctions" du C++, il est possible de remplacer les 3 premiers constructeurs de votre classe par un seul, utilisable de 3 manières différentes. Procéder à cette modification. Compiler et tester.

1. On aurait meilleur compte, dans la vraie vie, de programmer un *opérateur de conversion* vers double. Cf cours.

2 Encapsuler pour régner ?

En Mathématiques, le dénominateur d'un nombre rationnel ne peut pas être nul. Par exemple, la fraction $^3/_0$ n'a pas de sens. Pourtant, comme on vient de le voir, votre classe Rational permet encore ce genre de chose... Il convient donc que la classe garantisse que l'attribut dénominateur `denom` ne puisse jamais valoir `0` (c'est un "invariant de classe").

Grâce au principe d'encapsulation de la POO et aux mots clé de contrôle de visibilité, assurez-vous que cet invariant de classe est toujours respecté.

On considérera que toute tentative de créer une fraction avec un dénominateur nul est une "erreur grave". Dans ce cas, votre code affichera dans le terminal le texte : `"denom nul. Fin du programme."` puis sortira du programme (avec `exit(1);`). [2]

Compiler et tester (un programme de test `test_partie1_2.cpp` est fourni).

3 Des rationnels irréductibles

En mathématique, $^{18}/_{-30}$ et $^{-3}/_5$ sont deux représentations du *même* nombre rationnel. La seconde représentation est appelée la forme "irréductible".

On souhaite maintenant que nos objets rationnels soient toujours stockés en mémoire sous leur forme irréductible. Par exemple, le rationnel $^6/_{-10}$ devra être stocké sous la forme irréductible $^{-3}/_5$.

Pour rappel, pour obtenir la représentation irréductible d'un nombre rationnel n/d, il faut et suffit de diviser n et d par leur PGCD. Voici une fonction C qui calcule le PGCD de deux entiers passés en paramètre (disponible sur le site, fichier *calcul_pgcd.c*) :

```
// fonction calculer_pgcd()
// Calcule et retourne le pgcd de a et b.
// Si b est < 0, la valeur retournee sera negative.
// PRECONDITION : n2 non nul
int calculer_pgcd(int n1, int n2 ) {
   int r, a, b;
   a = abs(n1);
   b = abs(n2);
   while (b != 0) {
      r = a % b;
      a = b;
      b = r;
   }
   if(n2 < 0) {
      a = -a;
   }
   return a;
}
```

Dotez votre classe d'une méthode privée `void reduce()` qui modifie l'état (les valeurs des attributs) de `*this` pour que le rationnel soit sous forme irréductible. Modifiez votre classe pour appeler cette méthode à bon escient, de telle sorte que le nouvel invariant de classe "le rationnel est toujours stocké dans sa forme irréductible" soit toujours respecté.

Pourquoi est-il adéquat de déclarer cette méthode `void reduce()` privée ?

2. Nous verrons plus tard dans le cours comment gérer de façon plus fine de tels cas d'erreurs, au moyen du mécanisme des exceptions.

Écrire et tester un programme principal `main()` qui :
— Crée les 4 objets rationnels $r1 = {}^{0}/_{1}$ $r2 = {}^{3}/_{1}$ et $r3 = {}^{-9}/_{-6}$. et $r4 = {}^{9}/_{-6}$.
— Affiche leurs représentations textuelles dans le Terminal
— Affiche le numérateur de $r1$
— Affiche la valeur réelle approché de $r3$
— Modifie le dénominateur de $r2$ à la valeur 3 et affiche
— Modifie le numérateur de $r4$ à la valeur 6 et affiche
— Modifie le dénominateur de $r2$ à la valeur 0 et affiche.

4 Encore plus fort ?

On souhaite maintenant que l'utilisateur puisse choisir si son objet `Rational` est stocké sous forme irréductible ou non. Cette propriété doit pouvoir être décidée dès la création du rationnel, puis être modifiée une fois l'objet créé.

Comment faire ? Implanter votre solution.

Modifier le programme principal précédent :
— créer $r1$, $r2$ et $r3$ en "toujours irréductibles"
— créer $r4$ en forme "non réduite"
— après l'affectation du numérateur de $r4$, changer la propriété "irréductible" de $r4$ à `true`.
Un autre programme de test, `test_partie3_4.cpp`, est fourni.

5 Opérateur ami << d'écriture dans un flux

La méthode `afficher()` est sympathique, mais elle ne permet que d'afficher l'objet dans le Terminal. Or, il est utile de pouvoir écrire une représentation textuelle de l'état d'un objet dans plusieurs destinations : fichier texte, chaîne de caractère en mémoire, Terminal, etc.

La bonne manière d'y parvenir en C++ est de mettre en place l'opérateur ami `friend ostream& operator<<(ostream& out, const Rational & r)`, qui ajoute une représentation textuelle de `r` dans le flux `out` et retourne ce flux [3].

Commenter la méthode `afficher()`. Mettre en place l'opérateur ami `friend ostream& operator<<(ostream & out, const Rational & r)` pour la classe `Rational`. Modifier le programme principal en conséquence. Tester.

6 Un peu d'arithmétique...

Avec la surcharge d'opérateur en C++, il est possible de faire que la syntaxe usuelle des opérations arithmétiques s'applique sur des objets de type `Rational` : addition avec `+`, multiplication `+`, incrément avec `+=`, test d'égalité `==`, etc.

Dans cette partie, pour tester pas à pas vos fonctions, vous utiliserez le fichier `test_partie6.cpp` en commentant les appels des fonctionnalités pas encore implantées.

3. A partie de maintenant, toutes vos classes C++ seront dotées d'un tel opérateur ami `<<`.

6.1 Addition et multiplication de deux rationnels et opérateur puissance

Implanter les opérateurs amis de calcul arithmétique suivant :

— `friend Rational operator *(const Rational& a, const Rational& b)` qui réalise le produit des rationnels a et b. Pour rappel, la multiplication de deux rationnels est définie de la manière suivante : $\frac{n}{d} \times \frac{n'}{d'} = \frac{nn'}{dd'}$.
— `friend Rational operator +(const Rational& a, const Rational& b)` qui réalise la somme des rationnels a et b. Pour rappel, l'addition de deux rationnels est définie de la manière suivante : $\frac{n}{d} + \frac{n'}{d'} = \frac{nd'+n'd}{dd'}$.

Dans tous les cas, le rationnel retourné sera mis dans sa forme irréductible si et seulement si l'un au moins des rationnels a et b est lui même irréductible.

Implanter la fonction amie suivante :

— `friend Rational pow(const Rational& r, int n)` qui calcule et retourne l'élévation à la puissance n du rationnel r.

6.2 Addition "en place"

Implanter l'opérateur membre d'addition "en place" `void operator +=(const Rational& a)` qui affecte à `*this` à le résultat de `(*this) + r`. Son existence permet d'utiliser la syntaxe concise `r += r1;` à la place de `r = r + r1;`.

6.3 Comparons des rationnels

Pour pouvoir comparer des rationnels entre eux, implanter les opérateurs amis de comparaison suivants :

— `friend bool operator ==(const Rational& a, const Rational& b)` qui retourne "true" si `a` et `b` représentent le même nombre rationnel. On s'assurera que la notion d'égalité entre nombres rationnels est bien entendue comme en mathématiques. Ainsi, par exemple, $\frac{6}{4}$ sera considéré égal à $\frac{3}{2}$.
— `friend bool operator !=(const Rational& a, const Rational& b)` qui retourne "true" si `a` et `b` ne représentent pas le même nombre rationnel.

7 Comptons nos rationnels : attributs et membres de classe

On souhaite enfin doter la classe `Rational` d'une moyen de savoir, à tout instant, le nombre d'instances (d'objets de type `Rational`) qui existent en mémoire.

La notion de "nombre d'instances" est une propriété de la *classe* et non pas de ses instances ; on aura donc recours à des "membres (attributs et méthodes) de classe" (mot clé `static`).

— Doter la classe d'un nouvel attribut "de classe" entier `nbInstances`, initialisé à 0 ;
— Dans chaque constructeurs (c'est à dire : à chaque fois qu'un nouvel objet `Rational` est créé), incrémenter cet attribut ;
— Dans le destructeur (c'est à dire : à chaque fois qu'un nouvel objet `Rational` est détruit), décrémenter cet attribut ;
— Doter la classe d'une méthode "de classe" `int getNbInstances()` permettant d'accéder au nombre d'instances
— Tester au moyen du programme de test `test_partie7.cpp`