Sommaire

La Classe Ratio

Introduction

Dans ce TP, nous nous proposons de développer une classe C++ nommée `Ratio` permettant de traiter des nombres rationnels quelconques sans passer par des `float` ou des `double` qui nous obligeraient à commettre des erreurs d'arrondi.

Travail

En partant de ce qui a été vu en cours, compléter la classe `Ratio` en fonction du cahier des charges suivant. Toutes les fonctionnalités demandées doivent être réalisées, et leur bon fonctionnement vérifié.

La classe `Ratio` code un nombre rationnel de la forme n/d en stockant indépendamment son numérateur (n) et son dénominateur (d) qui sont tous les deux des entiers. Le numérateur servira à coder le signe du rationnel. Le dénominateur devra donc toujours rester positif. L'accès aux membres numérateur et dénominateur ne doit pas être permis autrement que par des fonctions accesseurs. Deux types d'accesseurs existeront : des accesseurs permettant d'extraire la valeur du numérateur (ou du dénominateur), et des accesseurs permettant la modification des membres.

Une méthode `reduire()` doit être intégrée à la classe. Elle permet de réduire la fraction : c'est-à-dire de trouver la fraction équivalente ayant les numérateurs et dénominateurs les plus petits possibles en valeur absolue. On rappelle que pour réduire la fraction, il faut diviser le numérateur et le dénominateur par leurs le PGCD (voir Liens). Par exemple, la fraction 45/30 n'est pas réduite car $45 = 5\times3\times3$ et $30=5\times3\times2$, on peut donc diviser le numérateur et le dénominateur par le PGCD(45,30)=15, et ainsi obtenir la fraction réduite 3/2.

La modification par un des accesseurs de modification doit déclencher systématiquement une simplification de la fraction, ainsi la fraction sera toujours réduite.

Il doit être possible de définir une instance de `Ratio` en précisant la valeur du numérateur et du dénominateur. Il doit également être possible de créer un rationnel à partir d'un nombre entier relatif quelconque ; on posera alors que le dénominateur vaut 1. Enfin, on doit pouvoir créer un rationnel nul lorsqu'on défini un `Ratio` sans préciser la valeur de son numérateur et de son dénominateur.

Comme pour toute classe que vous écrirez, vous devez définir le constructeur de copie pour un `Ratio`.

L'écriture d'un rationnel sur un flux (avec <<) doit produire une sortie de la forme "3/2" si le dénominateur est différent de 1, de la forme "10" lorsque le dénominateur est égal à 1 et qu'il s'agit donc d'un entier relatif.

La lecture sur un flux (avec >>) doit aussi lire un rationnel de la forme "-45/-12" aussi bien que "-23". La lecture, comme la modification par les accesseurs, devra provoquer une réduction de la fraction.

Il doit être possible d'affecter une instance de `Ratio` à une autre.

Il doit être possible de calculer une valeur approchée du rationnel avec une méthode `double valeur()`.

Il doit être possible d'inverser un rationnel (attention, il y a un petit piège à éviter avec le signe du rationnel).

Il soit être possible d'additionner deux `Ratio`, un `Ratio` et un entier relatif... ainsi que l'inverse, c'est-à-dire additionner un entier relatif avec un `Ratio`.

Sur le même principe, toutes les opérations possibles autour de la soustraction, de la multiplication et de la division doivent être réalisées.

Il doit également être possible d'utiliser les opérateurs de pré- et post-incrémentation, de pré- et post-décrémentation, ainsi que les opérateurs `+=`, `-=`, `*=`, et `/=`.

Remarques

- Avez-vous mis des `const` à tous les endroits où leur présence serait justifiée ?
- Avez-vous pris garde à ce que tout `Ratio` en mémoire soit toujours sous forme réduite ?

Si oui, tentez de répondre à la question subsidiaire...

Question subsidiaire

Coder une méthode `void Ratio::convert(double valeur, double precision)` pour trouver un rationnel approchant `valeur` avec une erreur au plus égale à `precision`. On se servira pour ce calcul d'une approximation par des fractions continues (dont vous pouvez trouver une définition sur Wikipedia (voir Liens) en anglais pour l'instant. Regardez en particulier la manière dont est obtenue un approximation de pi avec la fraction 355/113).

Ajouter le constructeur, permettant de créer un rationnel approchant un réel à une précision donnée.

Liens

- Page du cours de programmation C/C++ d'Éric REMY :

http://www.iut-arles.up.univ-mrs.fr/eremy/Ens/Info1.C++/index.html

- Planche de TP n°1 d'Éric REMY :

http://pluton.up.univ-mrs.fr/eremy/Ens/LPSIL.C++/tp1/index.html

- Pour approfondir vos connaissances en C et C++ :

http://www.iut-arles.up.univ-mrs.fr/eremy/Ens/Info1.C++/approfondir.html

- PGCD (Plus grand commun diviseur) :

http://fr.wikipedia.org/wiki/PGCD

- Approximation par fractions continues (Continued fraction) :

http://en.wikipedia.org/wiki/Continued_fraction

```
// X = x0 + 1 / (x1 + 1 / (x2 + 1 / ... + xn + 1)
// on pose :
// u0 = valeur
// x0 = [valeur]
// r0 = x0 / 1 et n0 = 1, d0 = 0
// on définit :
// u(n) = 1 / (u(n-1) - x(n-1))
// x(n) = [u(n)]
// r(n) = (r(n-1).num * x(n) + n(n-1)) / (r(n-1).den * x(n) + d(n-1))
// n(n) = r(n-1).num
// d(n) = r(n-1).den
```

Validation

```
int main() {
   Ratio r;

   cout << "-> r = Ratio() : " << endl;
   cout << "r = " << r << endl;

   cout << "-> r.getNum() : " << endl;
   cout << "r.num = " << r.getNum() << endl;

   cout << "-> r.getDen() : " << endl;
   cout << "r.den : " << r.getDen() << endl;

   cout << "-> r = Ratio(-5) : " << endl;
   r = Ratio(-5);
   cout << "r = " << r << endl;

   cout << "-> r = Ratio(4, -6) : " << endl;
   r = Ratio(4, -6);
   cout << "r = " << r << endl;

   cout << "-> r = Ratio(4, 0) : " << endl;
   try {
      r = Ratio(4, 0);
   }
   catch (exception & e) {
      cout << "Exception interceptée : " << e.what() << endl;
      r = 4;
   }
   cout << "r = " << r << endl;

   cout << "-> r.setDen(-6) : " << endl;
   r.setDen(-6);
   cout << "r = " << r << endl;

   cout << "-> r.setDen(0) : " << endl;
   r.setDen(0);
   cout << "r = " << r << endl;

   cout << "-> r.setNum(-6) : " << endl;
   r.setNum(-6);
   cout << "r = " << r << endl;

   cout << "-> r.inverser() : " << endl;
   r.inverser();
   cout << "r = " << r << endl;
```

```
cout << "-> r.setNum(0) : " << endl;
r.setNum(0);
cout << "r = " << r << endl;

cout << "-> r.inverser() : " << endl;
r.inverser();
cout << "r = " << r << endl;

cout << "-> Affectation de (30, 24) à r : " << endl;
r = Ratio(30, 24);
cout << "-r = " << -r << endl;

cout << "-> Ratio(1, 3) + Ratio(3, 2) : " << endl;
cout << (Ratio(1, 3) + Ratio(3, 2))  << endl;
cout << "-> Ratio(1, 3) + 1 : " << endl;
cout << (Ratio(1, 3) + 1)  << endl;
cout << "-> 1 + Ratio(1, 3) : " << endl;
cout << (1 + Ratio(1, 3))  << endl;

cout << "-> Ratio(11, 6) - Ratio(1, 6) : " << endl;
cout << (Ratio(11, 6) - Ratio(1, 6))  << endl;

cout << "-> Ratio(5, 3) * Ratio(3, 2) : " << endl;
cout << (Ratio(5, 3) * Ratio(3, 2))  << endl;

cout << "-> Ratio(15, 6) / Ratio(15, 6) : " << endl;
cout << (Ratio(15, 6) / Ratio(15, 6))  << endl;

cout << "-> r = Ratio(2, 5) : " << endl;
r = Ratio(2, 5);
cout << "r = " << r << endl;

cout << "-> r++ : " << endl;
cout << "r = " << r++ << endl;
cout << "r (apr->s) = " << r << endl;

cout << "-> ++r : " << endl;
cout << "r = " << ++r << endl;
cout << "r (apr->s) = " << r << endl;

cout << "-> r-- : " << endl;
cout << "r = " << r-- << endl;
cout << "r (apr->s) = " << r << endl;

cout << "-> --r : " << endl;
cout << "r = " << --r << endl;
cout << "r (apr->s) = " << r << endl;
```

```
    cout << "Entrez un ratio sous la forme x/y : ";
    cin >> r;
    if (! cin) {
       cout << "ERREUR de lecture\n";
       exit (-1);
    }
    cout << "r = " << r << endl;
    cout << "r (double) = " << r.valeur() << endl;
    if (cin.bad()) cout << "NON RATIO" << endl;

    r += 5;
    cout << "-> r += 5" << endl;
    cout << "r = " << r << endl;

    r -= 2;
    cout << "-> r -= 2" << endl;
    cout << "r = " << r << endl;

    r *= 2;
    cout << "-> r *= 2" << endl;
    cout << "r = " << r << endl;

    r /= 3;
    cout << "-> r /= 3" << endl;
    cout << "r = " << r << endl;

    cout << "-> Approximation de PI : " << endl;
    r = Ratio::convert(M_PI, 1e-10);
    cout.precision(10);
    cout << r << " = " << r.valeur() << endl;

    return 0;
}
```