TP C++: Serie 2

Exercice 1 : Surcharge des operateurs

Objective:
Definir le comportement des operateurs entre les objects de classe defini.
Les opérateurs qui peuvent être surchargés (38 symbol) : + - * / % ^ & | ~ ! = < > += -= *= /= %= ^= &= |= << >> >>= <<= == != <= >= <=> && || ++ -- , ->* -> ( ) [ ]
Les opérateurs qui ne peuvent pas être surchargés: (::) (Sizeof) (.) (*) (?:)
Syntaxe:

```
class MaClass
{   public:
      returnType operator symbol (arguments)
      { // instructions
      }
};
```

Énoncé:
L'objectif de cet exercice est de définir les opérateurs de comparaisons et les opérateurs arithmétiques d'une classe *Fraction* en utilisant les fonctions membres et les fonctions amies.

1. Créer la classe Fraction possédant deux données membres « num » et « den » qui correspondent respectivement au numérateur et au dénominateur de la fraction.
2. Définir un constructeur d'initialisation
3. Définir une fonction membre afficher() sous form de num / den.
4. Définir les opérateurs arithmétiques (+, - , * et /) entre deux fractions
5. Définir les opérateurs arithmétiques (+, - , * et /) entre une fraction et un nombre
6. Définir les opérateurs de comparaison (==, !=, <, >, <=, >=) sur deux fractions.
7. Créer un programme de test.

Exemple d'exécution 1 :

```
Fraction F1 : 1/2
Fraction F2 : 3/5
F1 + F2 = 11/10
F1 - F2 = -1/10
F1 * F2 = 3/10
F1 / F2 = 5/6
F1 + 5 = 11/2
3 * F2 = 9/5
Les deux fractions ne sont pas egales
La fraction la plus grande est : 3/5
```

Remarque: On ne demande pas de normaliser la fraction

Exercice 2 : Surcharge des fonctions

Objective:
Avoir le même nom d'une fonction mais avec des différents paramètres qui supporte des cas différents.

Syntaxe:

```
returnType fonctionName (argument1, argument2){
// k arguments
}
returnType fonctionName (argument1, argument2, argument3){
// n arguments avec n!=k
}
```

Énoncé:

1) Ecrire un programme à l'aide des fonctions (pas de classe), pour calculer la somme de deux ou trois entiers (Utiliser deux méthodes).

Exemple d'exécution :

```
Donner trois entiers: 3 9 8
Methode 1:
3 + 9 = 12
3 + 9 + 8 = 20
Methode 2:
3 + 9 = 12
3 + 9 + 8 = 20
```

2) Refaire la fonction somme sans valeur de retour. (Utiliser 4 méthodes)

3) Écrire une fonction qui convertit un nombre d'heures, de minutes et de secondes en un nombre de secondes équivalentes. Tester la fonction avec les exemples suivants :

Exemple 1 : Heure = 2, Minute = 39, Seconde = 45
Exemple 2 : Heure = 8, Minute = 26
Exemple 3 : Heure = 5
Exemple 4 : Heure = 3, Seconde = 19

Exemple d'exécution :

```
Exemple 1 : 2h 39min 45s = 9585 secondes
Exemple 2 : 8h 26min = 30360 secondes
Exemple 3 : 5h = 18000 secondes
Exemple 4 : 3h 19s = 10819 secondes
```

Remarque : Seul le paramètre Heure qui sera obligatoire.

Correction: https://github.com/hm43/ExercicesTpCpp.git