Programmation C++

Surcharge des opérateurs

ING2-GSI

CY Tech

2023-2024

Surcharge des opérateurs

Exemple : Fraction

```
#ifndef __FRACTION_HPP_
#define __FRACTION_HPP_

class Fraction {
private :
    int numerateur = 0;
    int denominateur = 1;
public :
    Fraction();
    Fraction(int n);
    Fraction(int n, int d);
    ~Fraction();
    // autres methodes
};

#endif
```

Des opérations sur les fractions

```
Fraction f1(1,8);
Fraction f2(1,2);
Fraction f3(1,5);
Fraction res;

//Beurk !
res =  Add(Add(Mul(f1,f2),Mul(f2,f3)),Mul(f3,f1));

//Carrement mieux !
res =  f1*f2 + f2*f3 + f3*f1 ;
```

Surcharge des opérateurs

- Les opérateurs ont tous le même préfixe : **operator**
  - arithmétique : `operator+, operator-, operator*, ...`
  - comparaison : `operator==, operator<, operator>=, ...`
  - racourcis : `operator+=, operator/=, operator++, ...`
  - autres : `operator<<, operator[], operator->, ...`
- Tous les opérateurs ne sont pas surchargeable ▸ Liste compl`ete - Wiki
- La priorité et l'associativité des opérateurs ne changent pas (standard)
- Les opérateurs peuvent être définis comme *membres* ou *non membres* d'une classe

Ex1 : Fonction membre : operator*=

```
/* Fraction.hpp    */
void operator*= (const Fraction& autre);

/* Fraction.cpp    */
void Fraction::operator*= (const Fraction& autre) {
    numerateur *= autre.numerateur;
    denominateur *= autre.denominateur;
}

/* main.cpp    */
Fraction f1(4,5);
Fraction f2(3,11);
f1*= f2;    // Fraction : 12/55
```

Ex2 : Fonction membre : operator*

```
/* Fraction.hpp */
Fraction operator*(const Fraction& autre) const;

/* Fraction.cpp */
Fraction Fraction::operator*(const Fraction& autre)
  const {
    return Fraction(numerateur*autre.numerateur,
        denominateur*autre.denominateur);
}

/* main.cpp */
Fraction f1(4,5);
Fraction f2(3,11);
Fraction res = f1*f2;    // Fraction : 12/55
```

Ex2 : Comment ça marche ?

```
/* main.cpp    */
Fraction f1(4,5);
Fraction f2(3,11);
Fraction res = f1*f2;     // Fraction : 12/55
// L'ecriture f1*f2 correspond a f1.operator*(f2)
res = f1*f2*res; // Ok :  (f1*f2)*res

res = f1*2; // ok : f1.operator*(2)
// Conversion de type : constructeur par parametre

res = 2*f1 // Erreur ! 2.operator*(f1)
// int : pas operator* avec une fraction en parametre
// pas de fonction non membre operator*(int,Fraction)
```

Ex3 : Fonction non membre : operator*

```
/* Fraction.hpp   : Declaree hors de la classe */
Fraction operator*(const Fraction& f1,
    const Fraction& f2);

/* Fraction.cpp    */
Fraction operator*(const Fraction& f1,
    const Fraction& f2) {
    return Fraction(f1.getNum()*f2.getNum(),
        f1.GetDen()*f2.GetDen());
}

/* main.cpp    */
Fraction f1(4,5);
Fraction res = 2*f1;    // Ok - Fraction : 8/5
```

Ex3 :operator* : Réutilisation ?

```
/* Fraction.cpp */
void Fraction::operator*= (const Fraction& f) {...}
Fraction& Fraction::operator= (const Fraction& f){...}

// Coder en fonction des autres operateurs
Fraction operator*(const Fraction& f1,
    const Fraction& f2) {
    Fraction res = f1;
    res *= f2;
    return (res);

}
```

Ex3 : operator* : Optimisation avec un scalaire ?

```
/* Fraction.cpp */

Fraction operator*(const Fraction& f1,
    const Fraction& f2) {
.../...
}

// Redefinir les operateurs de facon specifique
// afin d'eviter de creer l'objet Fraction
Fraction operator*(int n, const Fraction& f) {
.../...
}
```

Limites de la surcharge des opérateurs

- On ne peut surcharger que les opérateurs qui existent déjà
- On doit respecter l'arité de l'opérateur : binaire et unaire
  - On ne peut pas utiliser l'opérateur ++ pour réaliser l'opération a++b
  - On doit écrire (a++)+b ou a+(++b)
- Il n'y a pas de liens sémantiques implicites entre opérateurs redéfinis
  - a+=b n'est pas forcément la même opération que a=a+b
  - a==b n'implique pas que a!=b soit faux
- Il n'y a pas de commutativité automatique

"Whenever you can avoid friend functions, you should, because, much as in real life, friends are often more trouble than they're worth." Scott Meyers :-)