Analyse et Programmation Orientées Objets / C++

Surcharge des opérateurs

Surcharge des opérateurs (1)

➡ **Origine et justification**

- Améliorer la lisibilité des codes sources

- Faciliter l'usage de définitions génériques
- Etendre la notation de l'affectation aux objets
- Etendre les opérations arithmétiques aux objets
- Etendre la gestion des flux standard d'E/S aux fichiers

Surcharge des opérateurs (2)

➡ **Déclaration d'un opérateur de classe**

- Un opérateur est une méthode de la classe
- Tout opérateur est donc défini à partir d'une fonction
- Applicabilité de toutes les règles standards
- Usage du préfixe ***operator***

Surcharge des opérateurs (3)

➡ **Exemple d'extension des notations**

```
# include  "Point.h "
---
void main ( ) {
Point p1(1.0, 1.55, 3.7);
Point p2;

        p2=p1;
        cout << "Valeur du point P2 :   " << p2;
}
```

Déclaration d'un opérateur

➡ **Déclaration en langage C++**

```
class Point {
---
public :
Point ();
Point (double, double, double);

Point operator = (Point);

};
```

Usage de l'opérateur surchargé

➡ **Surcharge de l'opérateur =**

```
# include  "Point.h "
---
void main ( ) {
Point p1(1.0, 1.55, 3.7);
Point p2;

        p2=p1;          // p2.operator = (p1)

}
```

Définition d'un opérateur

➡ **Définition en langage C++**

```
# include  "Point.h "

Point Point::operator = (Point x) {
   m_X =  x.m_X;
   m_Y =  x.m_Y;
   m_Z =  x.m_Z;
   return *this;
}
```

Noter l 'usage de l 'opérateur . pour les attributs

Surcharge des opérateurs (4)

➡ **Limites d'applicabilité de la surcharge**

- Opérateur d'affectation =

- Opérateurs arithmétiques : +, -, *, /
- Opérateurs étendus : +=, -=, *=, /=
- Incrémentation/décrémentation : ++, --
- Opérateurs de comparaison : ==, !=, <, >, <=, >=
- Accès aux éléments des tableaux : []
- Gestion des E/S : <<, >>
- Allocation dynamique (new, delete)
- Opérateur fonctionnel : ( )

Surcharge des opérateurs (5)

➡ **Problèmes induits et solutions**

- Passage d'un objet par valeur
- **Recopie de l'objet** (structure des attributs) sur la pile
- Solution apportée par un constructeur de copie
- Problème des attributs dynamiques
- Impossibilité d'une solution générique C++
- Définition du constructeur de copie à la charge du programmeur si attributs dynamiques

Pour éviter le constructeur de copie

➡ **Déclaration en langage C++**

```
class Point {
---
public :
Point ();
Point (double, double, double);

Point& operator = (const Point&);
Point& operator = (Point*);

};
```

Cas d'un opérateur d 'E/S

➡ **Déclaration en langage C++**

```
class Point {
---
public :
Point ();
Point (double, double, double);

Point& operator = (const Point&);
friend ostream& operator << (ostream&, const Point&);
};
```

Définition d'un opérateur d 'E/S

➡ **Définition en langage C++**

```
# include  "Point.h "

ostream& operator << (ostream& cS, const Point& x) {
   cS <<"(";
   cS << x.m_X << ", ";
   cS << x.m_Y << ", ";
   cS << x.m_Z;
   cS << ")";
   return cS;
}
```

Usage de l'opérateur surchargé

➡ **Surcharge de l'opérateur <<**

```
# include  "Point.h "
---
void main ( ) {
Point p1(1.0, 1.55, 3.7);
---
        cout << p1;

}
```

Autres exemples de surcharge (1)

➡ **Opérateur + dans le corps des complexes**

```
# include   "RxR.h"

RxR* RxR::operator + (const RxR& z) {
double x= m_X+z.m_X;
double y= m_Y+z.m_Y;

      return new RxR(x, y);
}
```

Autres exemples de surcharge (2)

➡ **Opérateur += dans le corps des complexes**

```
# include   "RxR.h"

RxR& RxR::operator += (const RxR& z) {
    m_X +=  z.m_X;
    m_Y +=  z.m_Y;
    return *this;
}
```

Autres exemples de surcharge (3)

➡ **Opérateur == dans le corps des complexes**

```
# include   "RxR.h"

bool RxR::operator == (const RxR& z) {
   if (m_X !=  z.m_X) return false;
   if (m_Y !=  z.m_Y) return false;
   return true;
}
```