SURCHAGE D'OPÉRATEURS

INTRODUCTION À LA SURCHARGE D'OPÉRATEURS

DÉFINITION

La **surcharge d'opérateurs** en C++ permet de redéfinir le comportement des opérateurs existants pour manipuler des objets de **classes personnalisées**.

AVANTAGES

- **Améliore la lisibilité** du code en permettant d'utiliser des **opérateurs familiers**.
- **Simplifie la syntaxe** et rend le code plus **intuitif**.

BONNES PRATIQUES

- **Surchargez les opérateurs** uniquement lorsque cela améliore la clarté et la simplicité du code.
- Assurez-vous que la surcharge d'un opérateur est **logique** et conforme à l'utilisation habituelle de cet opérateur.
- Évitez de surcharger les opérateurs de manière incohérente avec leur **sémantique habituelle**.

SURCHARGE D'OPÉRATEURS UNAIRES

INTRODUCTION

Les **opérateurs unaires** sont des opérateurs qui agissent sur un seul **opérande**. En C++, il est possible de **surcharger** les opérateurs unaires pour personnaliser leur comportement lorsqu'ils sont utilisés avec des objets de nos propres classes.

Les opérateurs unaires peuvent être des opérateurs arithmétiques communs comme l'incrément (++) ou le décrément (--), ou d'autres opérateurs comme ! (négation logique) ou ~ (négation binaire).

OPÉRATEUR D'INCRÉMENTATION ET DE DÉCRÉMENTATION

SYNTAXE

```
class MaClasse {
  // Surcharge de l'opérateur d'**incrémentation préfixé**
  MaClasse& operator++();

  // Surcharge de l'opérateur d'**incrémentation postfixé**
  MaClasse operator++(int);

  // Surcharge de l'opérateur de **décrémentation préfixé**
  MaClasse& operator--();

  // Surcharge de l'opérateur de **décrémentation postfixé**
  MaClasse operator--(int);
};
```

Les opérateurs d'incrémentation et de décrémentation peuvent être surchargés pour définir des comportements spécifiques pour les instances d'une classe

EXEMPLES D'UTILISATION

```
MaClasse a;
++a; // Appelle l'**opérateur d'incrémentation préfixé**
a++; // Appelle l'**opérateur d'incrémentation postfixé**
--a; // Appelle l'**opérateur de décrémentation préfixé**
a--; // Appelle l'**opérateur de décrémentation postfixé**
```

Les opérateurs d'incrémentation et de décrémentation sont utilisés pour augmenter ou diminuer la valeur d'une variable. Les versions préfixées et postfixées ont un comportement légèrement différent en cas d'utilisation dans une expression.

OPÉRATEUR DE NÉGATION

SYNTAXE

```
class MaClasse {
  // Surcharge de l'opérateur de négation
  MaClasse operator-() const;
};
```

La surcharge d'opérateurs permet une syntaxe personnalisée pour les objets de nos classes en réutilisant des opérateurs existants, tels que +, -, *, /. De plus, il est possible de redéfinir les comportements des opérateurs en utilisant la surcharge.

EXEMPLES D'UTILISATION

```
MaClasse a;
MaClasse b = -a; // Appelle l'opérateur de négation
```

L'opérateur de **négation** permet de modifier la valeur d'un objet de la classe. C'est une fonction membre de la classe qui est définie avec l'opérateur `-`.

OPÉRATEUR D'INDIRECTION ET D'ADRESSE

SYNTAXE

```
class MaClasse {
  // Surcharge de l'**opérateur d'indirection**
  ValueType& operator*();

  // Surcharge de l'**opérateur d'adresse**
  MaClasse* operator&();
};
```

Cet exemple illustre la surcharge des opérateurs d'indirection et d'adresse pour personnaliser le comportement d'une classe.

EXEMPLES D'UTILISATION

```
MaClasse a;
ValueType valeur = *a; // Appelle l'opérateur d'**indirection**
MaClasse* ptr = &a; // Appelle l'opérateur d'**adresse**
```

Les opérateurs d'indirection et d'adresse sont très importants en C++ pour gérer les pointeurs et accéder aux valeurs stockées dans les adresses mémoire.

SURCHARGE D'OPÉRATEURS BINAIRES

OPÉRATEURS ARITHMÉTIQUES

SYNTAXE

Pour **surcharger** un opérateur arithmétique binaire, on utilise la syntaxe suivante :

```
class MyClass {
public:
    MyClass operator+(const MyClass& other) const;
    // autres opérateurs arithmétiques...
}
```

EXEMPLES D'UTILISATION

```
MyClass a, b;
MyClass c = a + b; // Utilise l'opérateur + surchargé
```

La surcharge d'opérateurs permet de redéfinir les opérations arithmétiques et relationnelles pour les objets d'une classe dans un langage orienté objet comme le C++.

OPÉRATEURS (+, -, *, /, %)

- + : **Addition**
- −: **Soustraction**
- * : **Multiplication**
- / : **Division**
- % : **Modulo**

OPÉRATEURS DE COMPARAISON

SYNTAXE

Pour surcharger un **opérateur de comparaison binaire**, on utilise cette syntaxe :

```
class MyClass {
public:
    bool operator==(const MyClass& other) const;
    bool operator!=(const MyClass& other) const;
    // autres opérateurs de comparaison...
};
```

La surcharge des opérateurs permet de définir comment les objets d'une classe spécifique réagissent à l'utilisation de différents opérateurs, tels que ==, !=, etc.

EXEMPLES D'UTILISATION

```
MyClass a, b;
bool isEqual = (a == b); // Utilise l'opérateur == surchargé
```

La surcharge d'opérateurs en C++ permet de redéfinir leur comportement pour des classes personnalisées.

OPÉRATEURS (==, !=, <, >, <=, >=)

- ==: **Égalité**
- !=: **Inégalité**
- <: **Inférieur**
- >: **Supérieur**
- <=: **Inférieur ou égal**
- >=: **Supérieur ou égal**

Ces opérateurs sont couramment utilisés dans les conditions (if, else if, etc.) pour déterminer le résultat d'une comparaison.

OPÉRATEURS D'AFFECTATION

SYNTAXE

Pour surcharger un opérateur d'affectation binaire, on utilise cette syntaxe :

```
class MyClass {
public:
    MyClass& operator=(const MyClass& other);
    MyClass& operator+=(const MyClass& other);
    // autres opérateurs d'affectation...
};
```

La surcharge des opérateurs permet d'utiliser des opérations prédéfinies (comme l'addition ou l'affectation) avec des objets de classes personnalisées.

EXEMPLES D'UTILISATION

```
MyClass a, b;
a = b; // Utilise l'opérateur = surchargé
a += b; // Utilise l'opérateur += surchargé
```

La surcharge d'opérateurs permet de redéfinir la fonctionnalité d'un opérateur existant pour un type de données particulier comme les objets.

OPÉRATEURS (=, +=, -=, *=, /=, %=)

- = : **Affectation**
- += : **Addition** et affectation
- -= : **Soustraction** et affectation
- *= : **Multiplication** et affectation
- /= : **Division** et affectation
- %= : **Modulo** et affectation

SURCHARGE D'OPÉRATEURS AVANCÉS

OPÉRATEURS DE FLUX D'ENTRÉE/SORTIE

Les **opérateurs de flux d'entrée et de sortie** (<< et >>) sont principalement utilisés avec les objets **cout** et **cin** pour l'affichage et la saisie de données.

SYNTAXE

```
ostream &operator<<(ostream &os, const MyClass &obj)
istream &operator>>(istream &is, MyClass &obj)
```

Ces opérateurs de flux permettent respectivement la surcharge de l'opérateur d'insertion (<<) pour l'écriture et de l'opérateur d'extraction (>>) pour la lecture, ce qui facilite l'utilisation avec des objets de type `MyClass`.

EXEMPLES D'UTILISATION

```
class MyClass {
    int number;

    friend ostream &operator<<(ostream &os, const MyClass &obj);
    friend istream &operator>>(istream &is, MyClass &obj);
}

ostream &operator<<(ostream &os, const MyClass &obj) {
    os << obj.number;
    return os;
}

istream &operator>>(istream &is, MyClass &obj) {
    is >> obj.number;
    return is;
}
```

OPÉRATEUR D'APPEL DE FONCTION (`operator()`)

Cet opérateur permet de transformer un objet en **fonction**. Il est utilisé pour **surcharger** le comportement de l'opérateur d'appel de fonction `()`.

L'utilisation de cet opérateur est principalement pour les langages de programmation orientée objet, où les objets peuvent être rendus "appelables" comme des fonctions.

SYNTAXE

```
ReturnType operator()(Arguments ...);
```

La syntaxe indiquée représente la signature d'un opérateur de fonction appel (ou opérateur parenthèses), utilisé pour surcharger les parenthèses et permettre d'utiliser un objet comme une fonction.

EXEMPLES D'UTILISATION

```
class Callable {
public:
    int operator()(int x) const {
        return x * x;
    }
}

Callable square;
int result = square(5); // result = 25
```

Cet exemple montre comment créer une classe avec une fonction de rappel en utilisant l'opérateur ().

OPÉRATEUR D'INDEXATION (`operator[]`)

Cet opérateur permet d'accéder aux éléments d'un objet comme s'il s'agissait d'un **tableau**.

Syntaxe	Description
objet[index]	Accès à la valeur de l'objet à l'index spécifié

L'opérateur `operator[]` peut être utilisé avec des objets tels que les tableaux, les chaînes de caractères et les dictionnaires.

SYNTAXE

```
ValueType &operator[](KeyType key);
const ValueType &operator[](KeyType key) const;
```

Ces deux fonctions permettent l'accès aux éléments d'un conteneur avec la clé "key". La première version est pour les conteneurs modifiables et la seconde pour les conteneurs non modifiables.

EXEMPLES D'UTILISATION

```
class Array {
    int data[10];

public:
    int &operator[](int index);
    const int &operator[](int index) const;
}

int &Array::operator[](int index) {
    return data[index];
}

const int &Array::operator[](int index) const {
    return data[index];
}
```

Cet exemple montre comment surcharger l'opérateur d'indexation [] pour une classe Array personnalisée en C++.