

**ISEN**

ALL IS DIGITAL!

LILLE



yncréa



# Le Langage C++

## Les constructeurs et les destructeurs

- Méthodes qui permet d'initialiser automatiquement une instance:
  - Toute instance créée statiquement
  - Ou dynamiquement avec l'opérateur new
- Un constructeur est une méthode qui :
  - A le même nom que la classe,
  - Ne retourne rien
- Il est possible d'avoir plusieurs constructeurs pour une même classe (surcharge).

MyClass.h:

```
class MyClass
{
private:

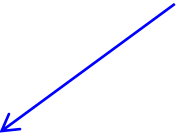
...

public:
    MyClass(); // Constructeur par défaut

    MyClass(type1 arg1,...); // Constructeurs personnalisés
    MyClass(type2 arg2,...);

};
```

Toujours (à quelques exceptions)



## Fraction.h

```
class Fraction
{
private:
    int num;
    int den;

public:
    ...
    Fraction();
    Fraction(int);
    Fraction(int, int);

};
```

- ✓ Bonnes habitudes
- ✓ Eviter les verifications à répétition

## Fraction.cpp

```
#include "Fraction.h"

Fraction::Fraction() {
    setNum(0);
    setDen(1);
}

Fraction::Fraction(int num) {
    setNum(num);
    setDen(1);
}

Fraction::Fraction(int num, int den) {
    setNum(num);
    setDen(den);
}

...
```

L'utilisation des arguments dans les constructeurs évite la redondance dans le code et permet de mixer les constructeurs personnalisés et le constructeur par défaut.

## Appel implicite

```
Fraction f1(8,3);  
Fraction* f3 = new Fraction();
```

## Appel explicite

```
Fraction f2 = Fraction(5);
```

```
Fraction::Fraction() {  
    setNum(0);  
    setDen(1);  
}
```

```
Fraction::Fraction(int num) {  
    setNum(num);  
    setDen(1);  
}
```

```
Fraction::Fraction(int num = 0, int den = 1) {  
    setNum(num);  
    setDen(den);  
}  
...
```

- Un constructeur par copie permet d'initialiser une instance à partir d'une autre instance (proprement).
- Il est appelé lorsque une nouvelle instance est créée à partir d'une autre:

```
Fraction f1;  
  
Fraction f2 = f1;           // implicitement  
Fraction f3 = Fraction(f1); // explicitement
```

MyClass.h:

```
class MyClass
{
...
public:
MyClass();           // constructeur par défaut
...
MyClass(type1 arg1,...); // constructeur personnalisé
MyClass(const MyClass&); // constructeur par copie
...
};
```

## LA REGLE D'OR

- Il est **OBLIGATOIRE** de coder un constructeur par copie lorsque la classe manipule des **attributs dynamiques** (des pointeurs).
- S'il n'est pas codé, le constructeur par copie a un comportement par défaut : il fait une copie attribut par attribut.



- Les destructeurs sont des méthodes qui permettent de désallouer les attributs d'une instance (la mémoire occupée) **proprement**.
- Un destructeur est appelé à chaque fois qu'une instance doit être détruite:
  - à la fin d'un bloc pour les variables déclarées de manière statiques.
  - Ou à l'appel de l'opérateur **delete**.
- Si aucun destructeur n'est écrit par le développeur, un **destructeur par défaut** est appliqué. Il libère la mémoire occupée par les différents attributs de l'instance.

Le destructeur de la classe **Person**  
est appelée pour l'instance **p**

```
void f()  
{  
    Person p("Smith");  
    ...  
}
```

Le destructeur de la classe **Person**  
est appelée pour l'instance dynamique **p**

```
void f()  
{  
    Person* p = new Person("Smith");  
    ...  
    delete p;  
    ...  
}
```

Le destructeur de la classe **Person**  
est appelée pour le tableau dynamique **p**

```
void f()  
{  
    Person* p = new Person[10];  
    ...  
    delete[] p;  
    ...  
}
```

- Un destructeur est une méthode de la classe :
  - Avec **le même nom** que la classe précédé du symbole `~`
  - **Aucun** argument
  - **Aucun** retour (même pas void)
- Avec ces conditions, on constate qu'il ne peut y avoir qu'**un seul destructeur** par classe. (à l'inverse des constructeurs qui peuvent être surchargés)

## MyClass.h

```
class MyClass
{
    ...
    ~MyClass(); // La definition du destructeur
    ...
};
```

## MyClass.cpp

```
MyClass::~~MyClass()
{
    ... // code du destructeur
}
```

## LA REGLE D'OR

Il est **OBLIGATOIRE** d'écrire un destructeur pour votre classe lorsque celle-ci manipule des attributs **allouées dynamiquement** (des pointeurs).