POLYTECH NICE-SOPHIA

Programmation multi-paradigmes en C++

Session #08 : *Move semantics & Rule of 3/5/0*

Julien Deantoni, Thomas Soucheyre,
Stephane Janel, Ken-Patrick
Lehrmann, Ludovic Tessier, Leandro
Fontoura Cupertino, Kevin Yeung

amadeus

POLYTECH NICE-SOPHIA

Agenda

Session 8: Move semantics & Rule of 3/5/0

01 Les membres spéciaux

02 *Move semantics*

1.

Les membres spéciaux
(Special members)

Les membres spéciaux

Ce sont ces fonctions membres qui peuvent être générées automatiquement par le compilateur

Les membres spéciaux

Ce sont ces fonctions membres qui peuvent être générées automatiquement par le compilateur

Vous en connaissez 4 :

- Le constructeur par défaut **X();**
- Le destructeur **~X();**
- Le constructeur par copie **X(const X &);**
- L'affectation par copie **X &operator=(const X &);**

Les membres spéciaux

Ils peuvent être :

- Déclarés explicitement par le développeur (*user declared*)
- Déclarés implicitement (par le compilateur)
- Non déclarés

Les membres spéciaux

Ils peuvent être :

- Déclarés explicitement par le développeur (*user declared*)
- Déclarés implicitement (par le compilateur)
- Non déclarés

Les membres spéciaux non déclarés ne participent pas à la résolution de surcharge (*overload resolution*)

Les membres spéciaux

Ils peuvent être :

- Déclarés explicitement par le développeur (*user declared*)
- Déclarés implicitement (par le compilateur)
- Non déclarés

Les membres spéciaux non déclarés ne participent pas à la résolution de surcharge (overload resolution)

Ils peuvent aussi être :

- defaulted ou deleted (explicitement ou implicitement)

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment
Nothing	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted
Copy constructor	not declared	defaulted	user declared	defaulted
Copy assignment	defaulted	defaulted	defaulted	user declared

Les membres spéciaux

compiler implicitly defines

user declares	Default constructor	Destructor	Copy constructor	Copy assignment
Nothing	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted
Copy constructor	not declared	defaulted	user declared	defaulted
Copy assignment	defaulted	defaulted	defaulted	user declared

N'importe quel constructeur déclaré explicitement empêche la génération implicite du constructeur par défaut

Les opérations de copie

Les opérations de copie déclarées implicitement par le compilateur se contentent de copier un à un chaque membre de la classe

Les opérations de copie

Les opérations de copie déclarées implicitement par le compilateur se contentent de copier un à un chaque membre de la classe

```
class X {
private:
    int _i;
    char _c;
};
```

```
X(const X &other) :
_i(other._i),
_c(other._c) {}
```

Les opérations de copie

Les opérations de copie déclarées implicitement par le compilateur se contentent de copier un à un chaque membre de la classe

```
class X {
private:
    int *_p;
};
```

```
X(const X &other) :
    _p(other._p) {}
```

Shallow copy vs. deep copy

```
class X {
public:
    X(int i) : _p(new int(i)) {}

private:
    int *_p;
};
```

```
X(const X &other) : _p(other._p) {}
```

Shallow copy
L'instance copiée pointe vers le même entier

```
X(const X &other) : _p(new int(*other._p)) {}
```

Deep copy
Chaque instance pointe vers un objet différent

Rule of 3

Si le développeur d'une classe a besoin de définir explicitement soit le destructeur, soit le constructeur par copie, soit l'affectation par copie, alors il a très certainement besoin de définir les 3

Rule of 3 : Mauvais exemple

```
class X {
public:
    X(int i) : _p(new int(i)) {}
    ~X() { delete _p; }

private:
    int *_p;
};
```

Rule of 3 : Mauvais exemple

```
class X {
public:
    X(int i) : _p(new int(i)) {}
    ~X() { delete _p; }

private:
    int *_p;
};
```

```
int main() {
    X x1(42);
    X x2 = x1;
};
```

free(): double free detected
Program terminated with signal: SIGSEGV

Rule of 3 : Bon exemple

```
class X {
public:
    X(int i) : _p(new int(i)) {}
    ~X() { delete _p; }

    X(const X &other) : _p(new int(*other._p)) {}
    X &operator=(const X &other) {
        if (this != &other) {
            delete _p;
            _p = new int(*other._p);
        }
        return *this;
    }

private:
    int *_p;
};
```

POLYTECH NICE-SOPHIA

2.

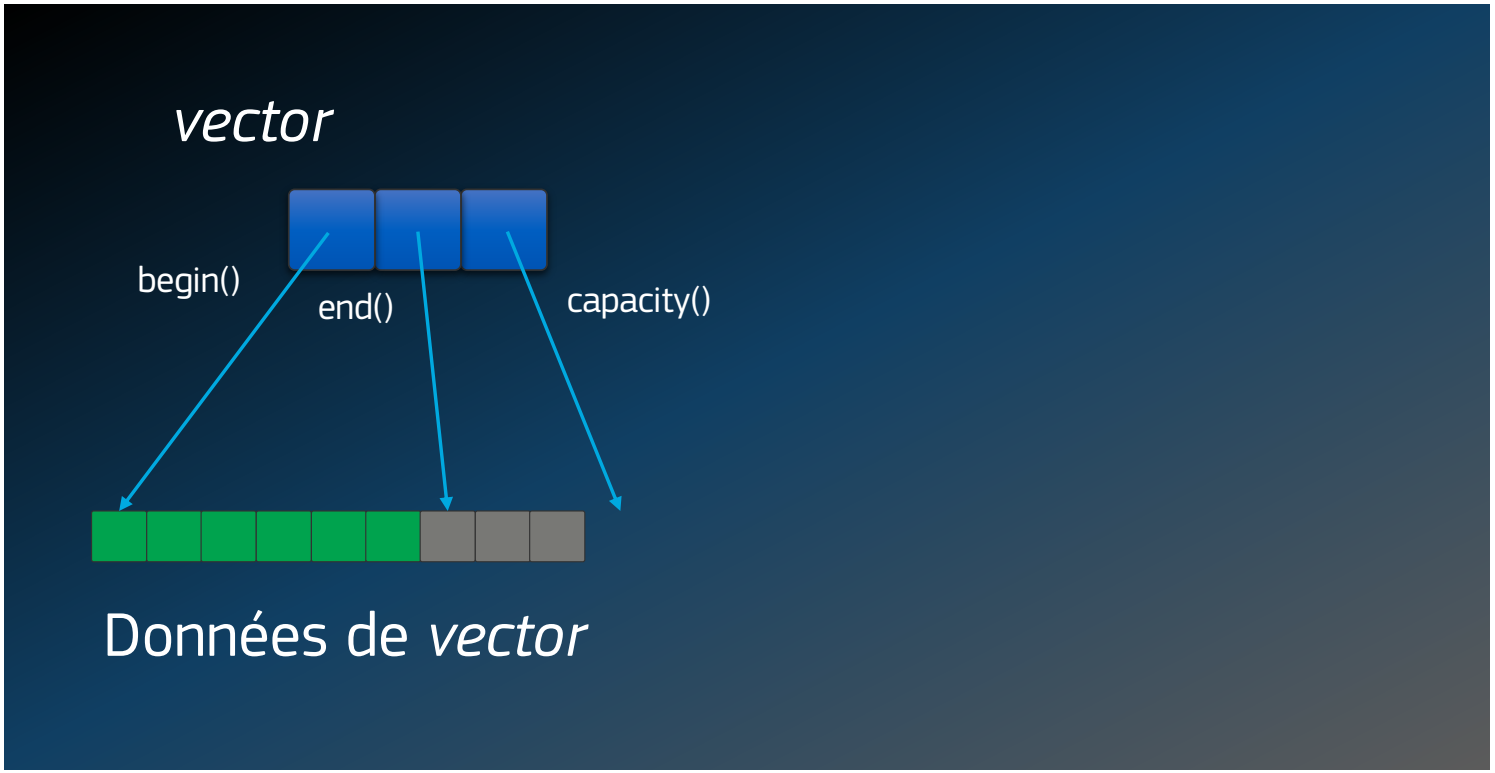
Move semantics

amadeus

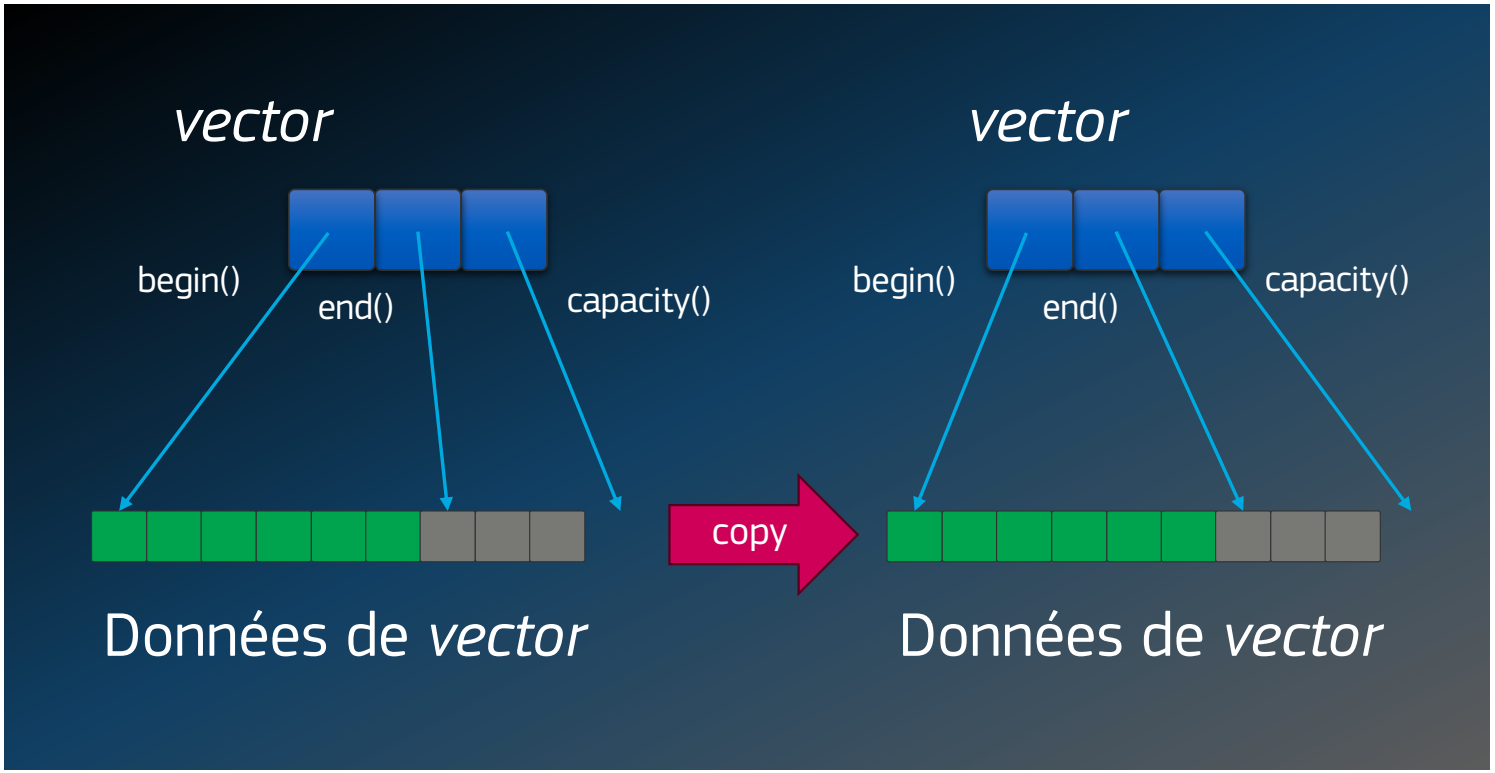
Move semantics

A l'origine, c'était juste une manière d'optimiser std::vector<T>

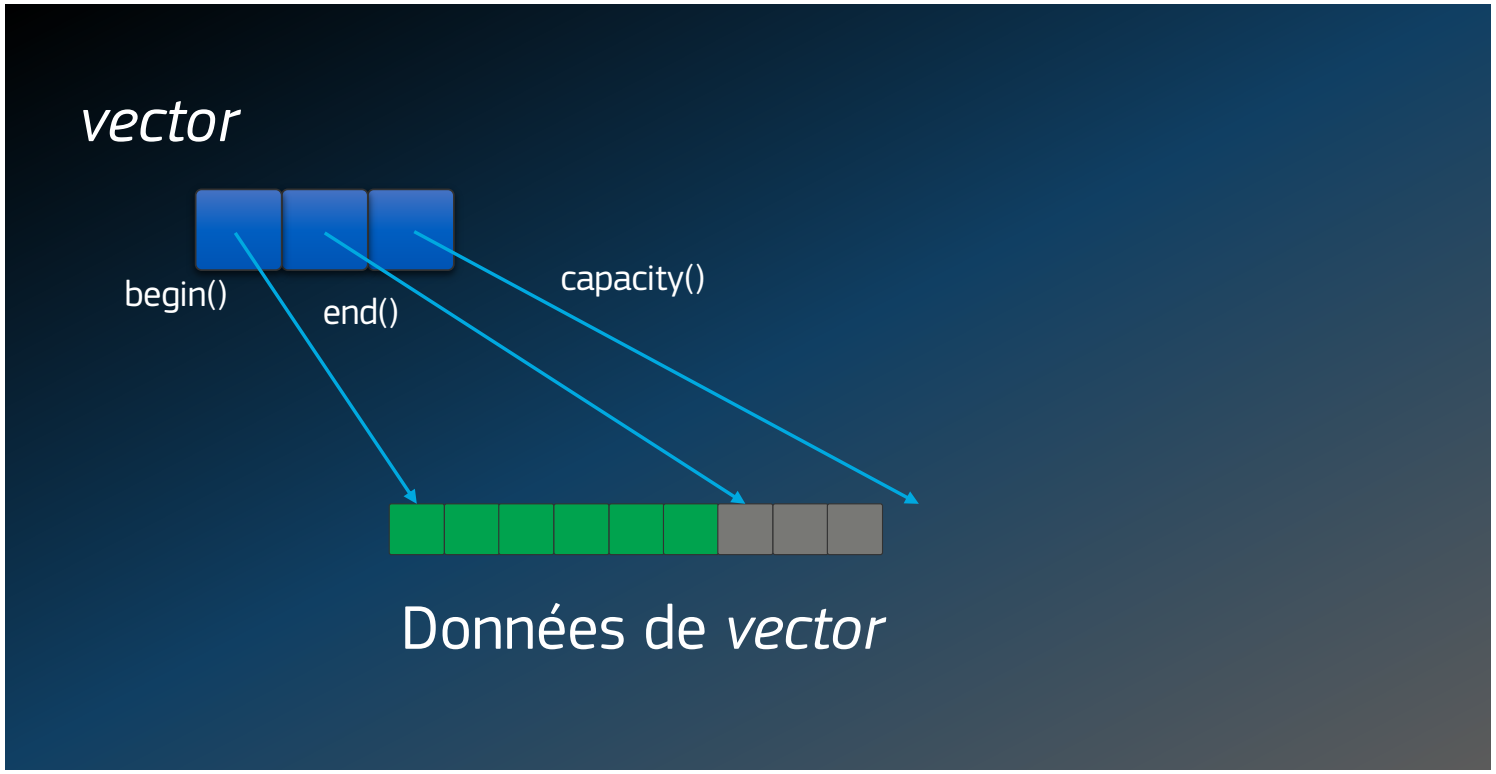
Fonctionnement de `std::vector`



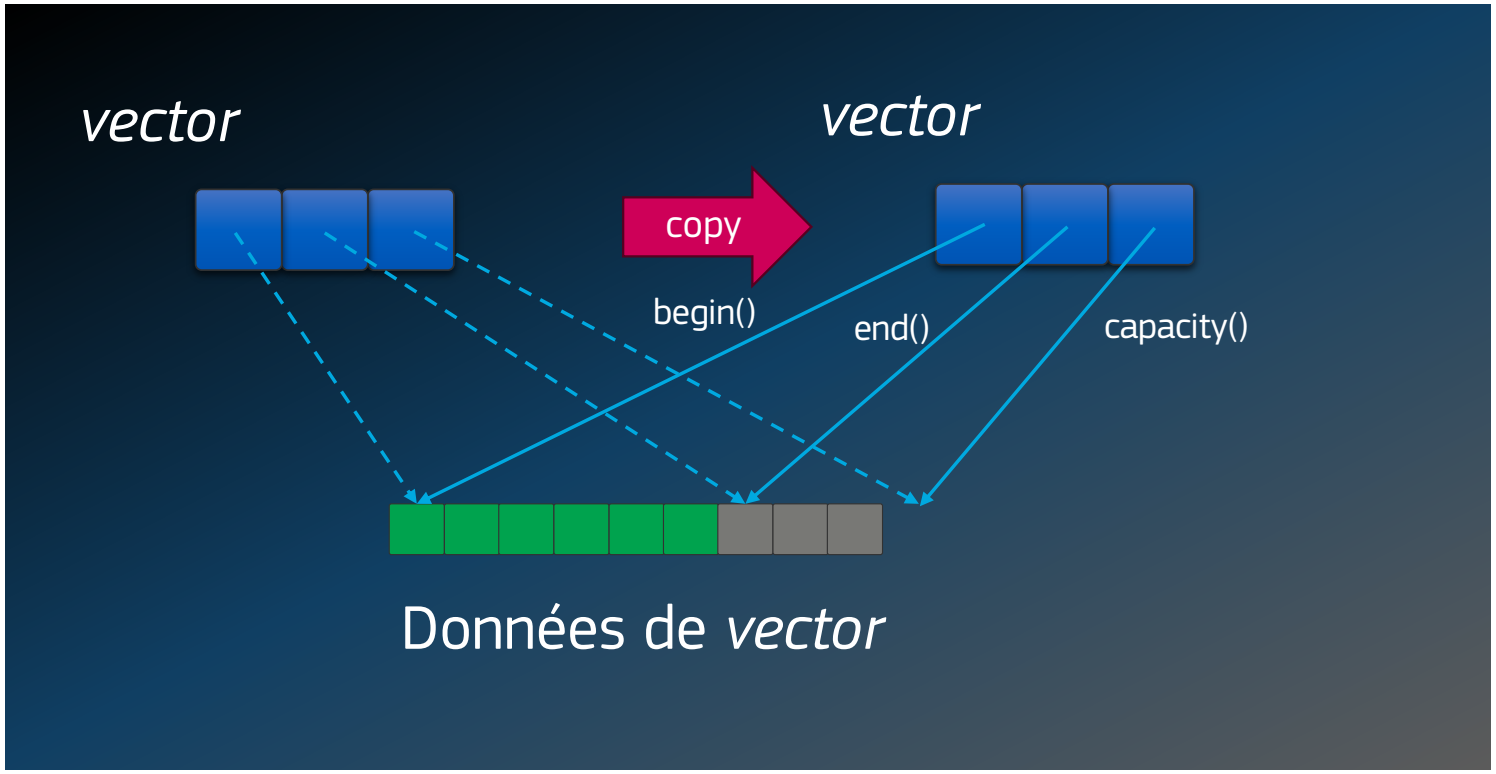
Copie d'un vecteur (*Copy*)



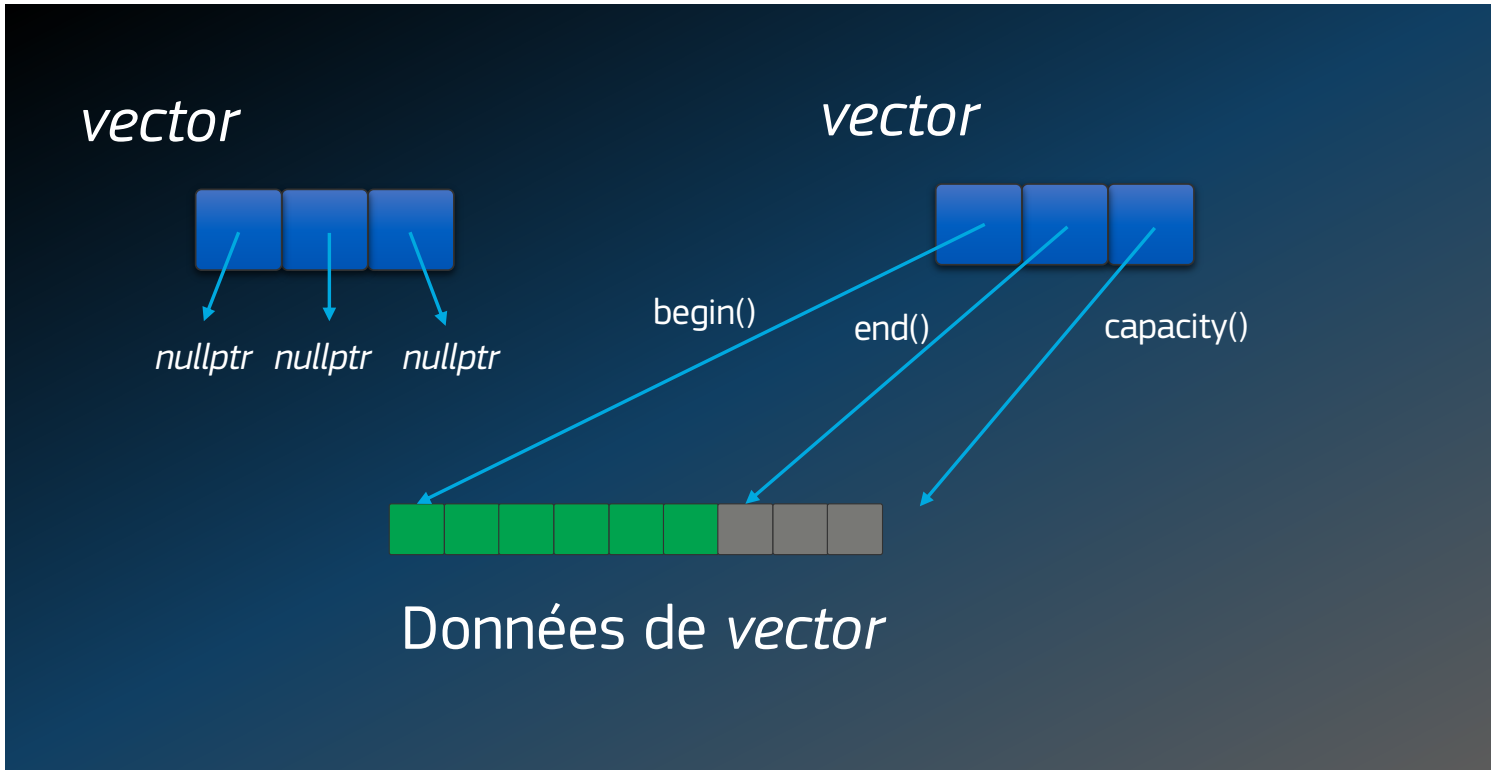
Déplacement d'un vecteur (*move*)



Déplacement d'un vecteur (*move*)



Déplacement d'un vecteur (*move*)



Les membres spéciaux

Il y en a maintenant 6 :

- Le constructeur par défaut **X();**
- Le destructeur **~X();**
- Le constructeur par copie **X(const X &);**
- L'affectation par copie **X &operator=(const X &);**
- Le constructeur par déplacement **X(X &&);**
- L'affectation par déplacement **X &operator=(X &&);**

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared

Les membres spéciaux

compiler implicitly defines

user declares	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
Copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
Copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
Copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
Copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
Copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
Move constructor	not declared	defaulted	deleted	deleted	user declared	not declared

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
Copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
Copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
Move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
Move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Les membres spéciaux

```
class X {
 public:
  X() = default;
  ~X() = default;
  X(const X &) = default;
  X &operator=(const X &) = default;
  X(X &&) = default;
  X &operator=(X &&) = default;
};
```

Les membres spéciaux

```
class X {
 public:
  X() = default;
  ~X() = default;
  X(const X &) = default;
  X &operator=(const X &) = default;
  X(X &&) = default;
  X &operator=(X &&) = default;
};
```

Les membres spéciaux

```
class X {
 public:
  X() = default;
  ~X() = default;
  X(const X &) = default;
  X &operator=(const X &) = default;



};
```

Les membres spéciaux

```
class X {
 public:

  ~X() = default;
  X(const X &) = default;
  X &operator=(const X &) = default;


};
```

Les membres spéciaux

```
class X {
 public:
  X() = default;
  ~X() = default;
  X(const X &) = default;
  X &operator=(const X &) = default;


};
```

Les membres spéciaux

```
class X {
 public:

  ~X() = default;
  X(const X &) = delete;
  X &operator=(const X &) = delete;
  X(X &&) = default;

};
```

Les membres spéciaux

```
class X {
 public:
  X() = default;
  ~X() = default;
  X(const X &) = delete;
  X &operator=(const X &) = delete;

  X &operator=(X &&) = default;
};
```

Les membres spéciaux

compiler implicitly defines

user declares

	Default constructor	Destructor	Copy constructor	Copy assignment	Move constructor	Move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
Default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
Destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
Copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
Copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
Move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
Move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Rule of 5

Si le développeur d'une classe a besoin de définir explicitement :

- soit le destructeur,
- soit le constructeur par copie,
- soit l'affectation par copie,
- soit le constructeur par déplacement,
- soit l'affectation par déplacement,

Alors il a très certainement besoin de définir les 5

Rule of 5 – Exemple

```
class X {
public:
     X(int i) : _p(new int(i)) {}
     ~X() { delete _p; }

     X(const X &other) : _p(new int(*other._p)) {}
     X &operator=(const X &other) {
          if (this != &other) {
               delete _p;
               _p = new int(*other._p);
          }
          return *this;
     }

     X(X &&other) : _p(other._p) { other._p = nullptr; }
     X &operator=(X &&other) {
          if (this != &other) {
               _p = other._p;
               other._p = nullptr;
          }
          return *this;
     }

private:
     int *_p;
};
```

Rule of 0

Le mieux est de ne définir aucun membre spécial (autre que le constructeur par défaut) et de laisser le compilateur gérer

Rule of 0 – Exemple

```
class X {
public:
     X(int i) : _p(std::make_shared<int>(i)) {}

private:
     std::shared_ptr<int> _p;
};
```

POLYTECH NICE-SOPHIA

Fin de la session

amadeus