

Programmation orientée objet

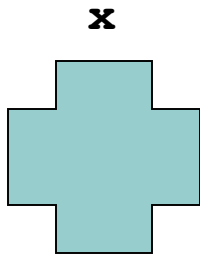
Passage de paramètres

Deux méthodes de passage de paramètre

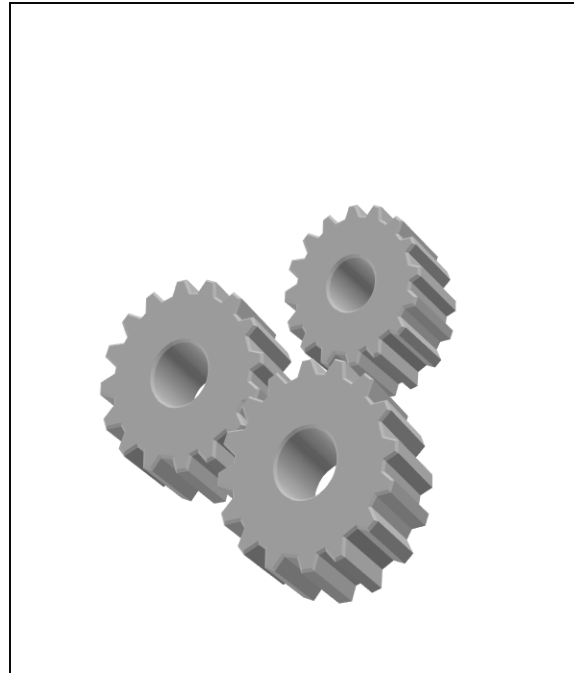
- ***Passage par valeur:*** on fait une copie du paramètre et c' est cette copie qui sera utilisée à l' intérieur de la fonction
- ***Passage par référence:*** on passe une référence à une entité, c' est-à-dire que l' entité passée en paramètre est manipulée directement, mais sous un autre nom

Passage par valeur

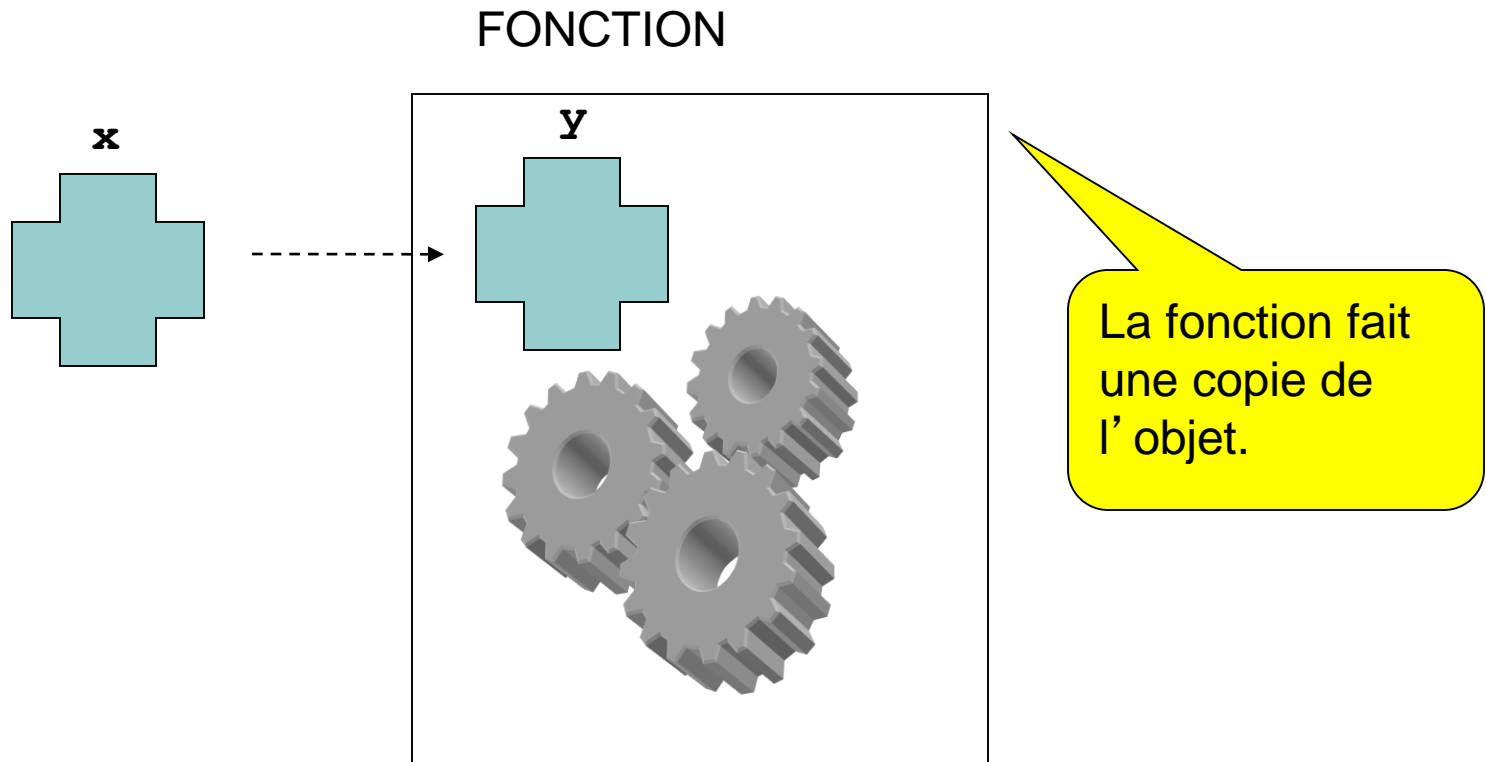
FONCTION



Un objet est
passé en
paramètre.

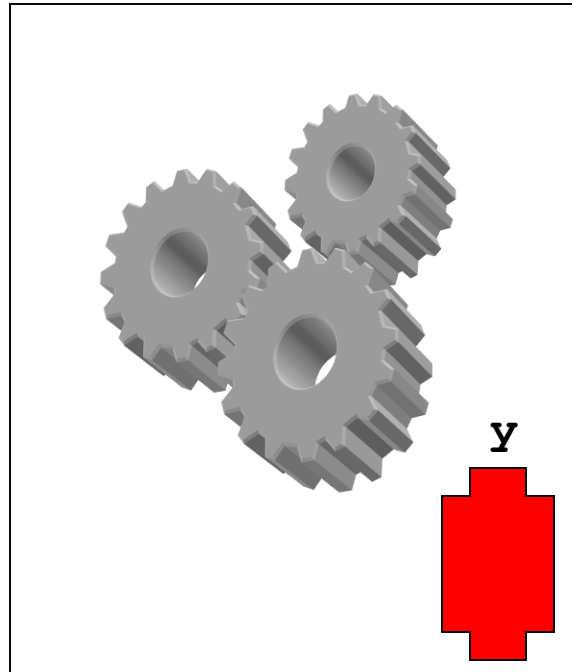
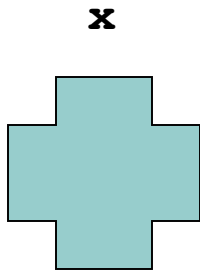


Passage par valeur



Passage par valeur

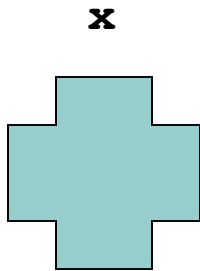
FONCTION



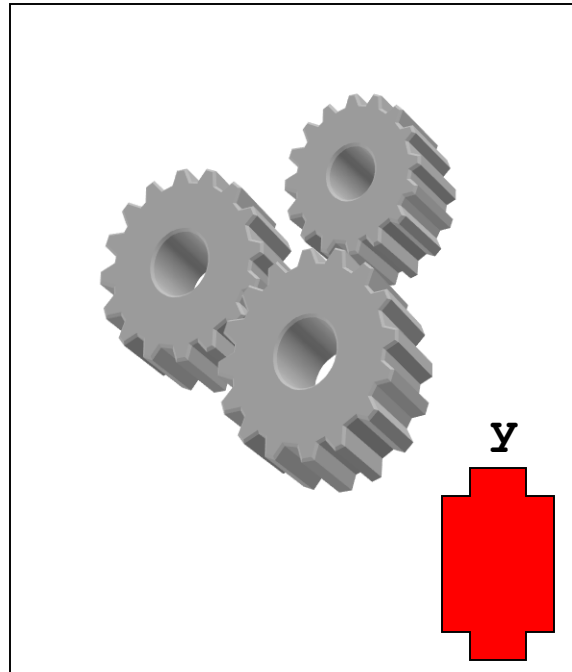
La fonction manipule l'objet et peut aussi changer son état.

Passage par valeur

FONCTION

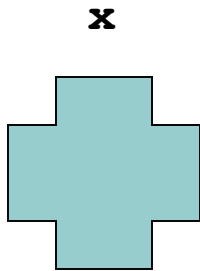


L'objet initial est
demeuré inchangé.

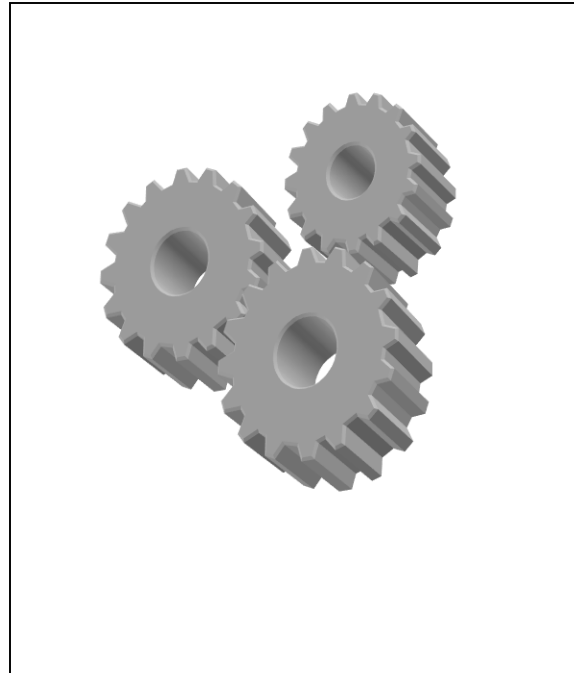


Passage par référence

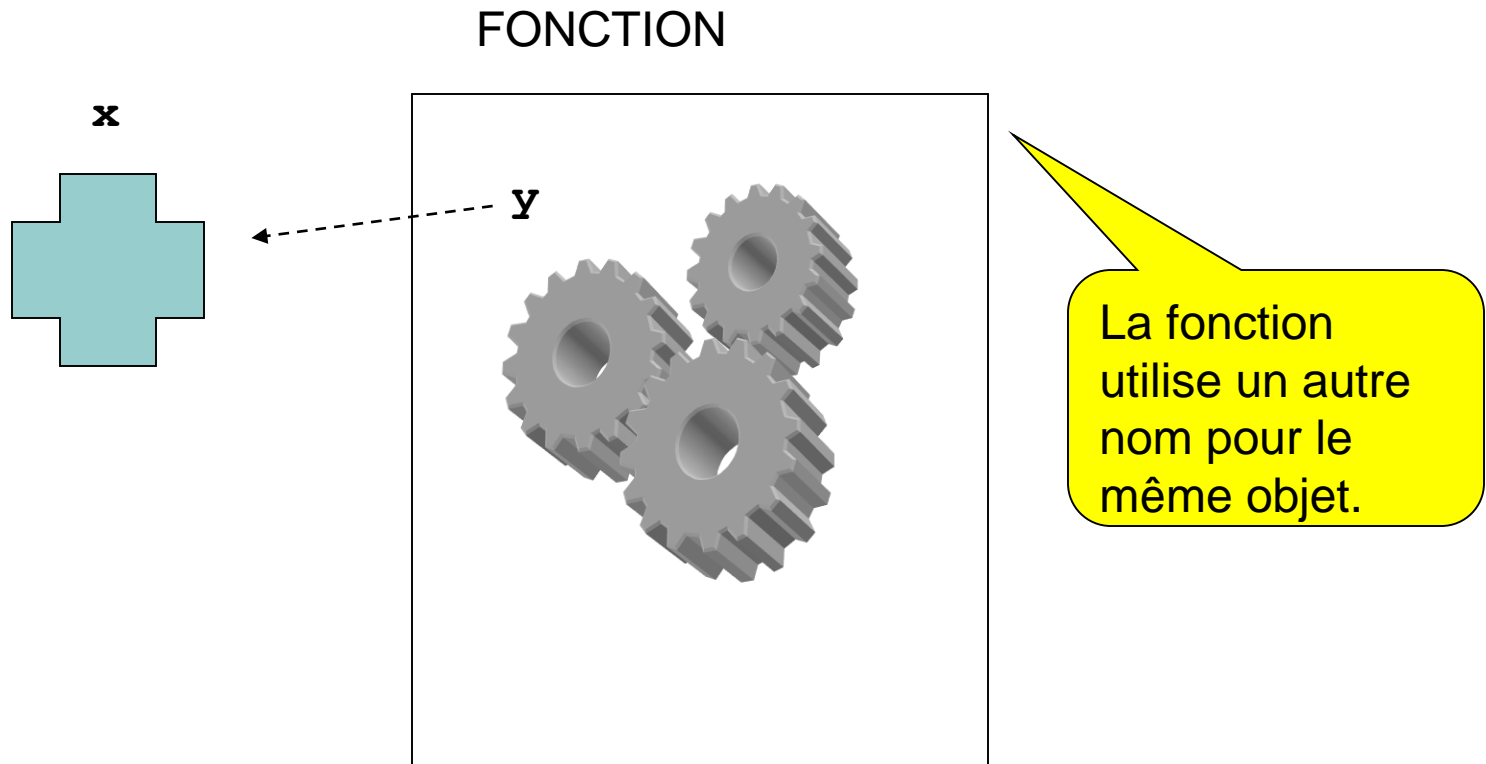
FONCTION



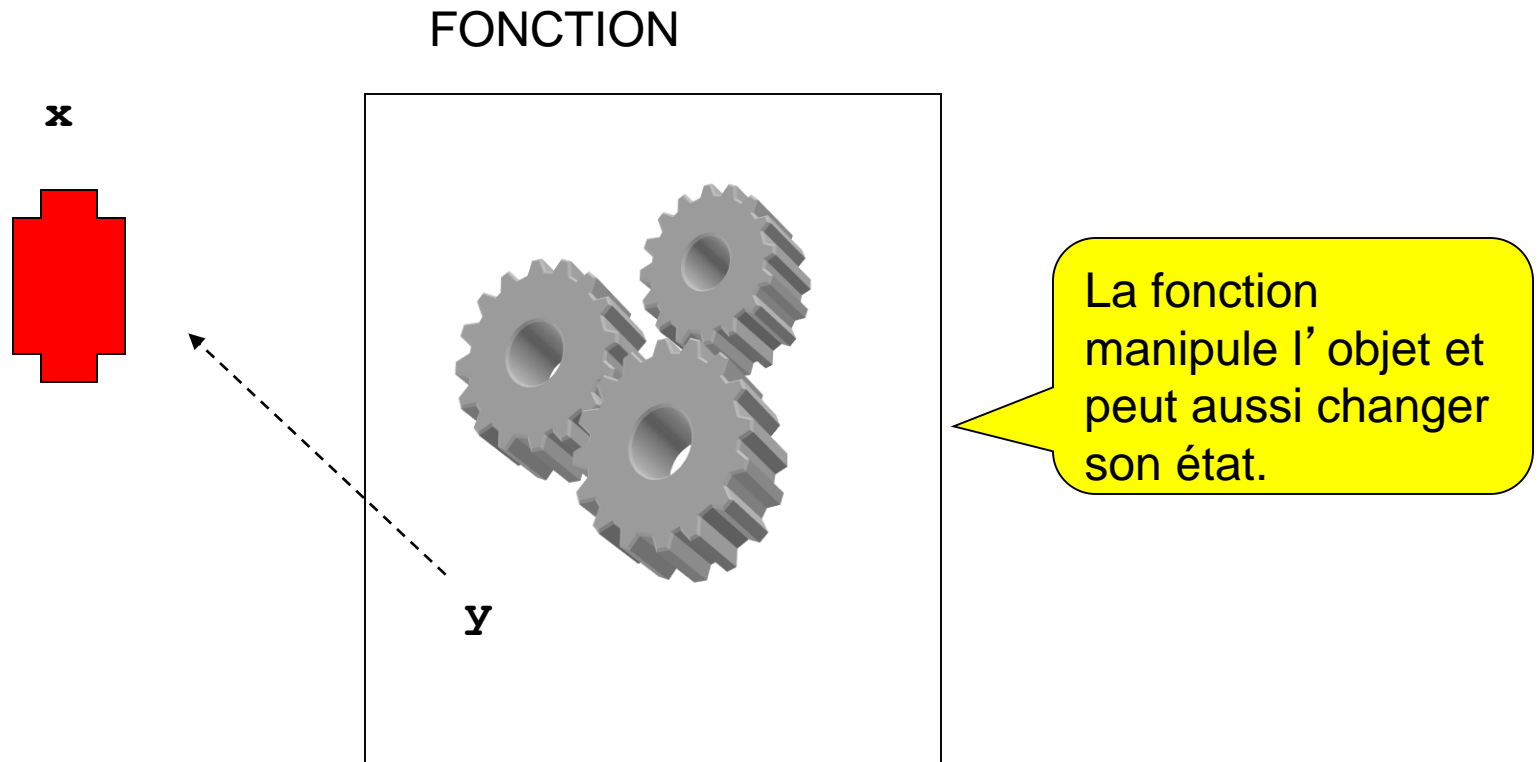
Un objet est
passé en
paramètre.



Passage par référence



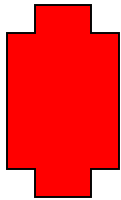
Passage par référence



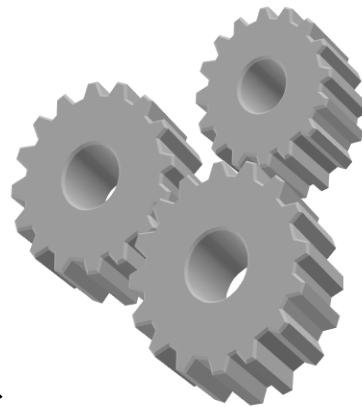
Passage par référence

FONCTION

x



L'objet initial peut
avoir changé.



y

Exemple problématique

```
void increase(Employee employe, double percentage)
{
    double newSalary= employe.getSalary() *
                      (1 + percentage/100);
    employe.setSalary(newSalary);
}
```

```
int main()
{
    Employee michel("Michel",100);
    increase(michel,5);
    cout << michel.getSalary();
    ...
}
```

Désolé, mais
le salaire n'a
pas changé!

Exemple corrigé

```
void increase(Employee& employe, double percentage)
{
    double newSalary= employe.getSalary() *
                      (1 + percentage/100);
    employe.setSalary(newSalary);
}
```

```
int main()
{
    Employee michel("Michel",100);
    increase(michel,5);
    cout << michel.getSalary();
    ...
}
```

employe est une référence au même objet que celui contenu dans la variable **michel**.

Le salaire aura finalement été augmenté!

Référence constante

- Souvent, on passe un objet par référence non pas parce qu'on veut le modifier, mais plutôt parce qu'on veut **éviter une copie qui est coûteuse**
- Pour éviter que cet objet soit modifié, on utilisera alors une référence constante

Référence constante (exemple)

```
void printCompany(const Company& c)
{
    cout << "Company " << c.getName();
    if (c.hasEmployees()) {
        cout << " has " << c.getNumberEmployees() << " employee(s) : "
                << endl;
        for (int i = 0; i < c.getNumberEmployees(); i++) {
            cout << " - Employee " << (i+1) << ": " <<
                c.getEmployeeByPos(i).getName() << endl;
        }
    } else {
        cout << " doesn't have any employee"
    }
}
```

Le compilateur permettra seulement l'utilisation de méthodes qui ont été déclarées const.