

Programmation orientée objet

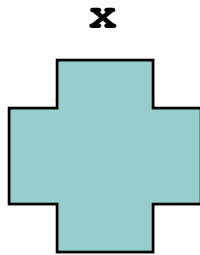
Passage de paramètres

Deux méthodes de passage de paramètre

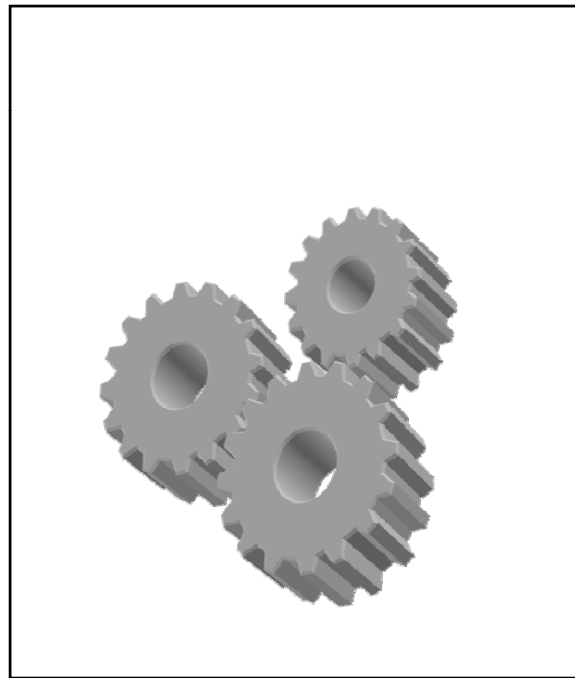
- *Passage par valeur*: on fait une copie du paramètre et c'est cette copie qui sera utilisée à l'intérieur de la fonction
- *Passage par référence*: on passe une référence à une entité, c'est-à-dire que l'entité passée en paramètre est manipulée directement, mais sous un autre nom

Passage par valeur

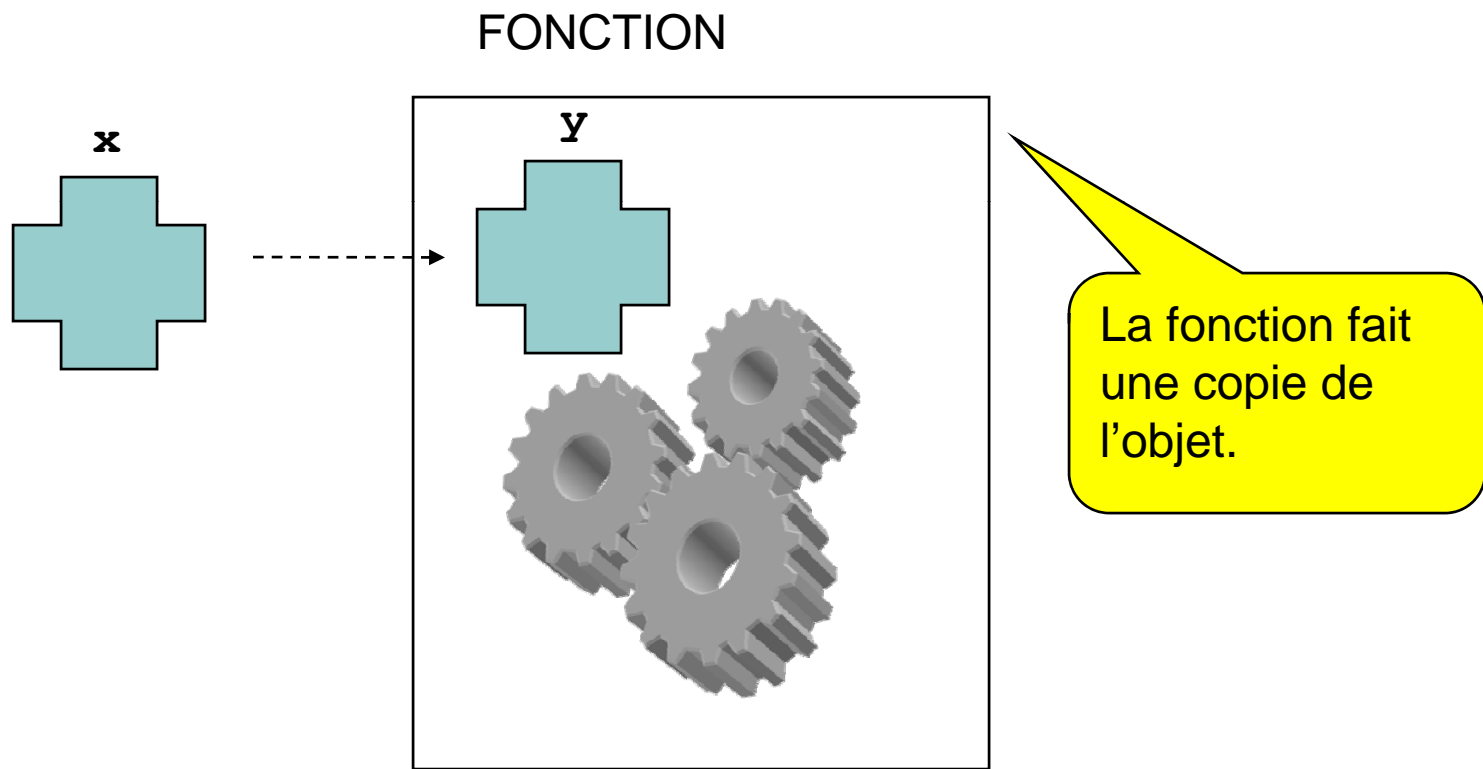
FONCTION



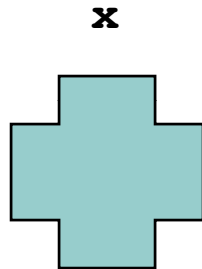
Un objet est
passé en
paramètre.



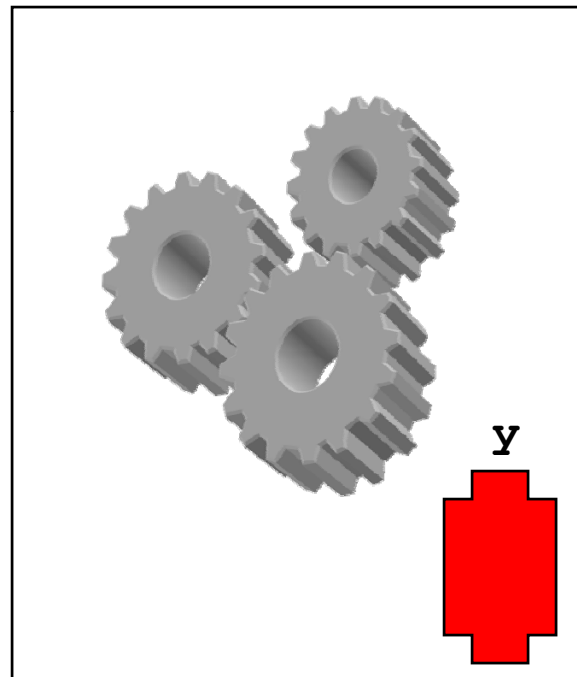
Passage par valeur




Passage par valeur



FONCTION

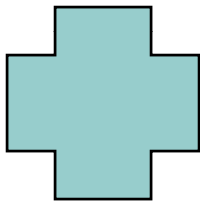


La fonction manipule l'objet et peut aussi changer son état. 

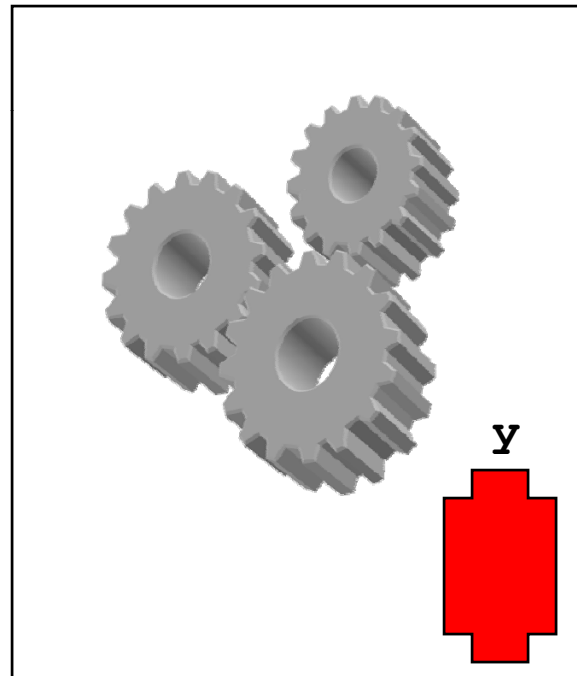
Passage par valeur

FONCTION

x



L'objet initial est
demeuré inchangé.

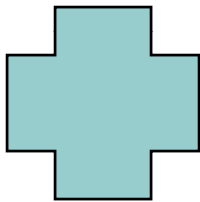


y

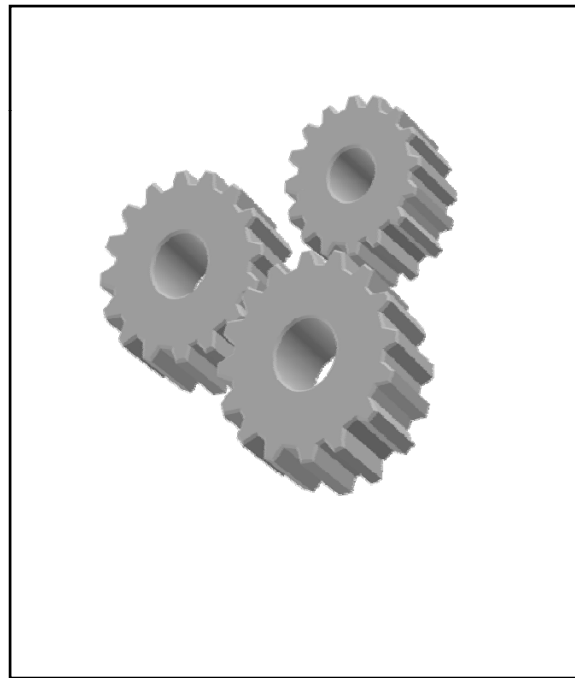
Passage par référence

FONCTION

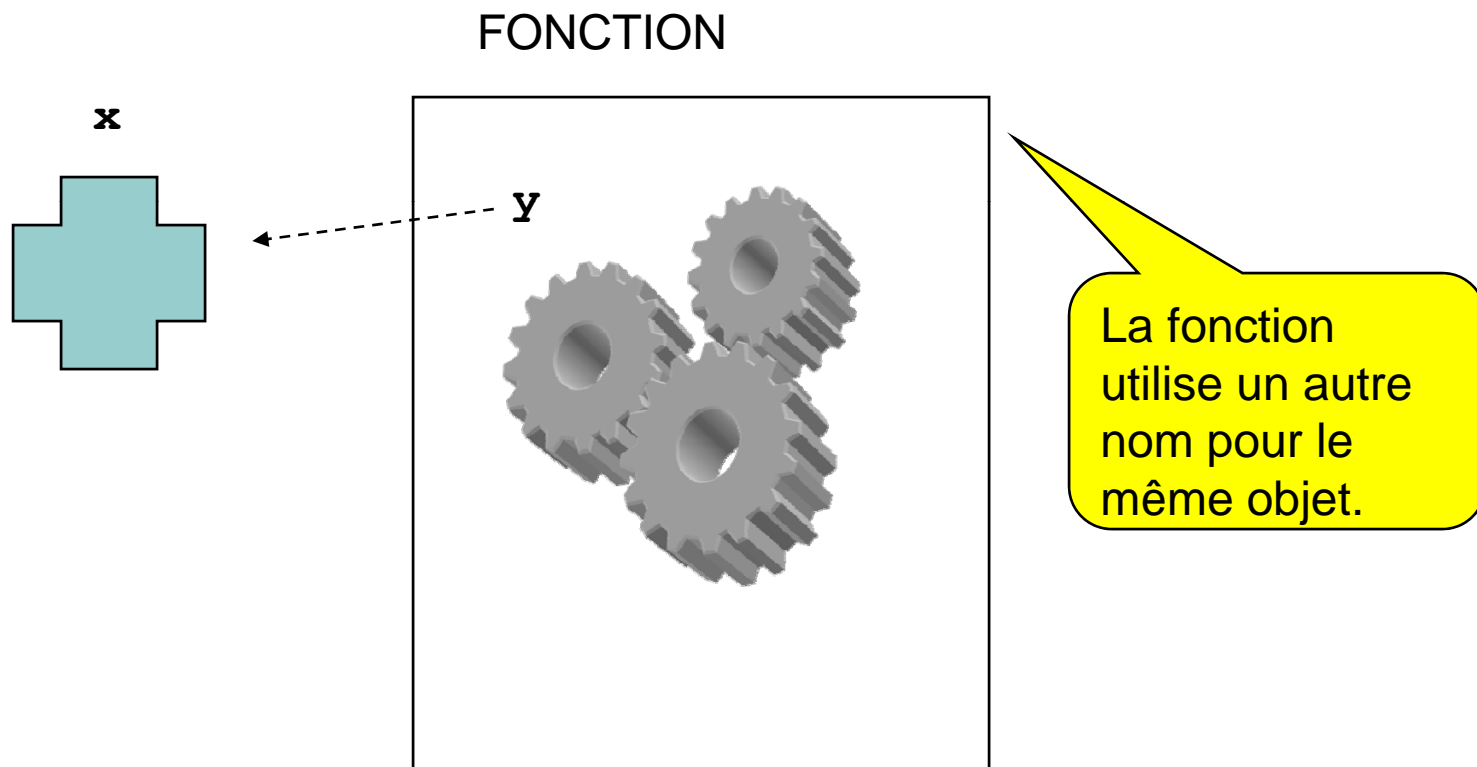
x



Un objet est
passé en
paramètre.

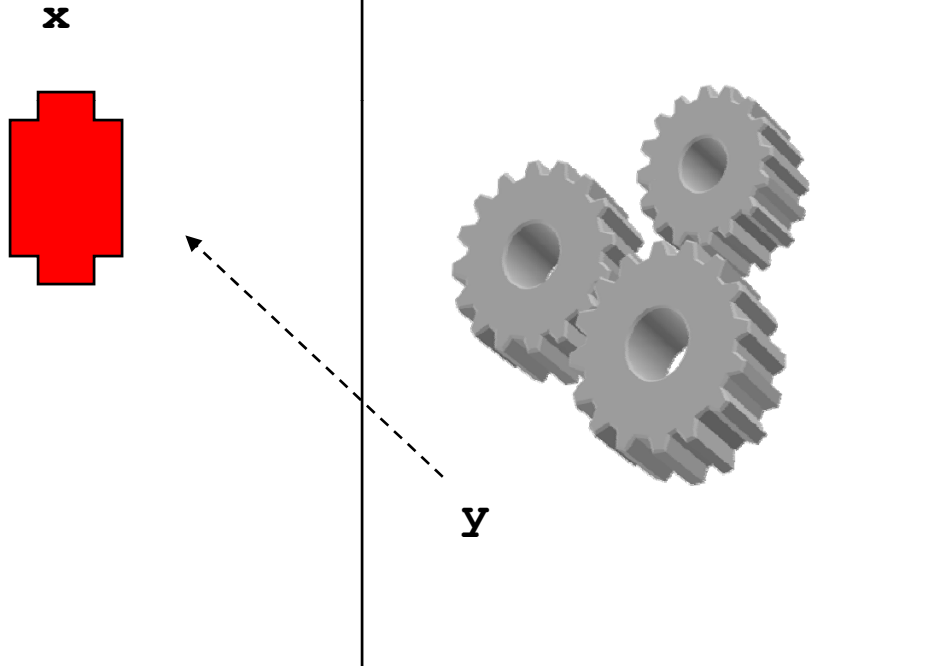


Passage par référence



Passage par référence

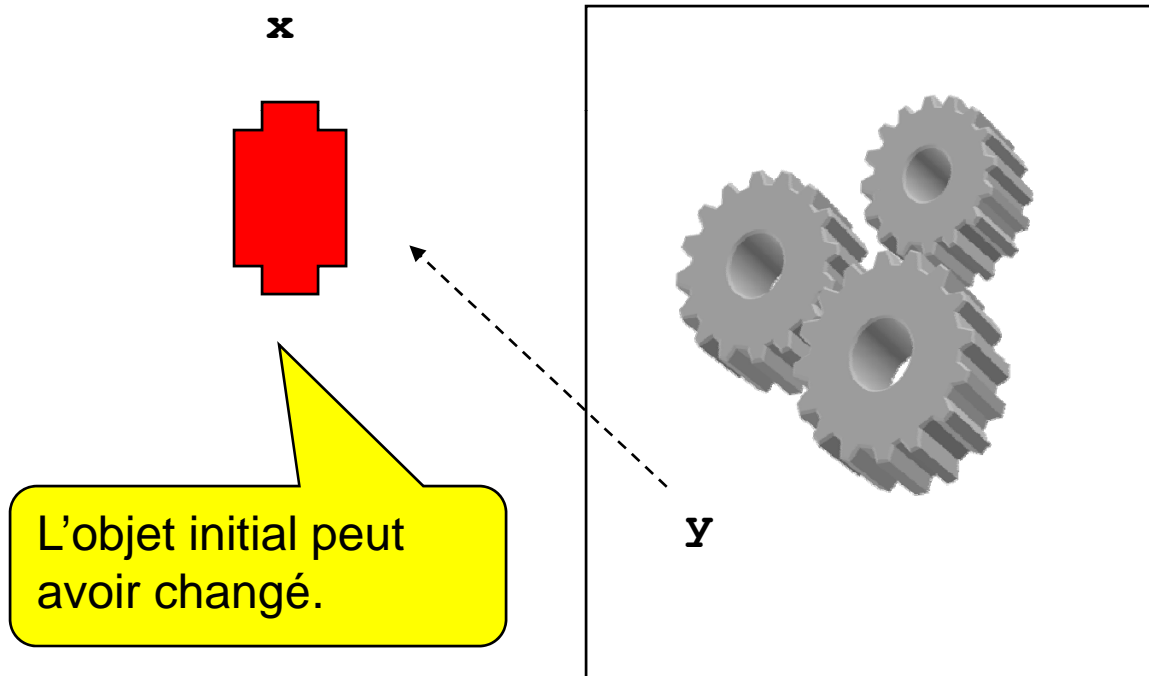
FONCTION



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

Passage par référence

FONCTION



Exemple problématique



```
void augmentation(Employe employe, double pourcentage)
{
    double nouveauSalaire = employe.getSalaire() *
                          (1 + pourcentage/100);
    employe.setSalaire(nouveauSalaire);
}

int main()
{
    Employe michel;
    augmentation(michel, 5);
    cout << michel.getSalaire();
    ...
}
```

Désolé, mais
le salaire n'a
pas changé!

Exemple corrigé



```
void augmentation(Employee& employe, double pourcentage)
{
    double nouveauSalaire = employe.getSalaire() *
                          (1 + pourcentage/100);
    employe.setSalaire(nouveauSalaire);
}

int main()
{
    Employee michel;
    augmentation(michel, 5);
    cout << michel.getSalaire();
    ...
}
```

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

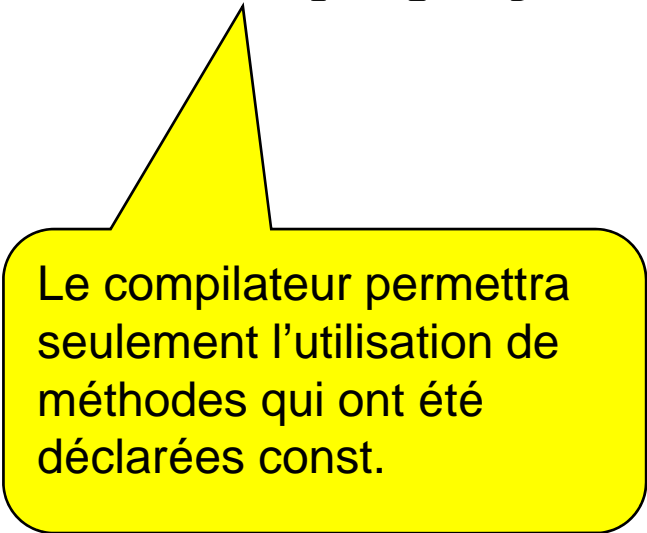
Le salaire aura finalement été augmenté!

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 imprimerEmploye(const Employe& employe)
{
    cout << "Nom : " << employe.getNom()
        << " Salaire : " << employe.getSalaire()
        << endl;
}
```



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

Valeurs par défaut

- La classe Employe a deux constructeurs:

```
class Employe
{
public:
    Employe();
    Employe(string nomEmploye, double salaireInitial);
    ...
};
```

- Le constructeur par défaut met la chaîne vide et la valeur 0 aux attributs nom et salaire, respectivement

Valeurs par défaut (suite)

- Étant donné que le constructeur par paramètres prend les valeurs fournies et les copie dans les attributs, on peut se contenter de celui-ci si on définit des valeurs par défaut aux paramètres:



```
class Employe
{
public:
    Employe(string nomEmploye = "",
            double salaireInitial = 0);
    ...
};
```


Valeurs par défaut (suite)

- En fait, c'est comme si nous avions maintenant trois constructeurs:

```
int main()
{
    Employe anonyme;
    Employe marie("Marie");
    Employe paul("Paulo",45000);
    ...
};
```