

Chapitre 13

Fonctions virtuelles et classes abstraites

1. Généralités

Ligature statique => choix de la fonction membre dépend du type statique par exemple de l'objet receveur.
Le typage statique est le type par défaut en C++.

Ligature dynamique => choix de la fonction dépend du type dynamique.

Le mot clé `virtual` force la ligature dynamique.

2. Fonction virtuelle (`virtual`)

```
#include <iostream>

using namespace std;

class X {
public:
    void f() { cout << "x:f\n"; }
};

class Y:public X {
public:
    void f() {cout << "y:f\n"; }
};
```

```
int main() {
    X a;
    Y b;

    X* ptr = &a;
    ptr->f();
    ptr = &b;
    ptr->f();
    return 0;
}
```

Sortie:

```
x:f
x:f
```

Suite à 2 appels `ptr->f()`, nous constatons que les deux utilisent la même version de la fonction `f` définie dans la classe de base `X`, car `ptr` est un pointeur du type `X` et ne voit donc que les propriétés de la classe `X`.

Comment faire pour que `ptr` fasse correctement l'appel à la fonction membre `f`?

Il faut déclarer la fonction `f` comme étant virtuelle.

```
class X {
public:
    virtual void f() { cout << "x:f\n"; }
};
```

Par cette écriture, le 2^e appel `ptr->f()` fera référence à la fonction `f` de la classe `Y`, au lieu de la fonction `f` de la classe de `X`.

3. Polymorphisme

Le même appel `ptr->f()` correspond à deux résultats différents (fonctions différentes: `f` de `X`, et `f` de `Y`).

La fonction est sélectionnée en fonction de la classe pointée par le pointeur `ptr`.

4. Ligature dynamique

L'association de l'appel à la partie du code à exécuter est différée (attend l'exécution pour définir le type) au moment de l'exécution du programme, opération plus coûteuse.

`virtual` permet de masquer le typage statique.

5. Qui peut être `virtual`, et qui ne le peut pas?

- Peuvent être `virtual`:
- Fonctions membres non statiques,
- Destructeurs.

Ne peuvent pas être virtual:

Champs membres,
Constructeurs.

6. Classes abstraites

- Une classe abstraite n'existe que pour être héritée.

- Une classe est dite abstraite si elle contient au moins une fonction virtuelle pure.

```
class X {  
  
    // Affiche est une fonction virtuelle pure, car = 0.  
    virtual void affiche() = 0;  
  
};
```

- Il est impossible de créer (instancier) un objet à partir d'une classe abstraite.

```
int main () {  
    X a; // Erreur  
    return 0;  
}
```

- Les classes qui héritent d'une classe abstraite doivent obligatoirement définir la ou les fonctions virtuelles pures.

```
class Y:public X {  
  
    // Même si le mot-clé virtual ne précède pas le nom de la fonction affiche,  
    // elle reste quand même virtuelle, car dans la classe de base, elle est déclarée ainsi.  
    // Donc nous n'avons pas besoin de le préciser encore une fois.  
    void affiche() {  
        cout << "Y:F\n";  
    }  
};
```