Modélisation objet& programmation objet en C++

TELLEZ Bruno, IUT Lyon1.

Département Informatique, Site de Bourg-en-Bresse

2023-2024

Le C++, un langage objet

Qu'est ce que C++ ?

- C++ repose sur le langage C
- C++ intègre les concepts orienté-objet
- C++ permet la programmation générique

Puissant mais plus complexe

Oubliez vos habitudes de programmation

C++ par rapport au C

- Données et Algorithmes
- C : Approche procédurale
  - Orientée algorithme
  - Décomposition en tâches (descendante)
  - Fonctions contraintes par le type des données
  - Sémantique des données

C++ par rapport au C

- C++ : Approche objet
  - Orienté donnée
  - Conception ascendante
  - Adaptation du langage aux problèmes : classes
  - Cohérence des données : cycle de vie de l'objet

Avantages du C++

- Réutilisabilité
- Encapsulation
  - Cohérence sémantique des données
  - Parties privé et public de l'objet
- Polymorphisme
  - Définitions multiples d'une fonction
  - Définitions multiples d'un opérateur
- Héritage
  - Création de nouvelles classes à partir d'anciennes
- Programmation générique
  - Création de structures indépendantes du type

Historique du C++

- Qui : Bjarne Stroustrup
- Quand : dans les années 1980
- Où : Laboratoire Bell
- Comment : En ajoutant des composantes objet au langage C
- Pourquoi : Pour être efficace plutôt que "pur objet" (proche du système, d'UNIX, etc.)

- C++ est un sur-ensemble du langage C
- Un programme C est valide pour un compilateur C++

Un programme C++

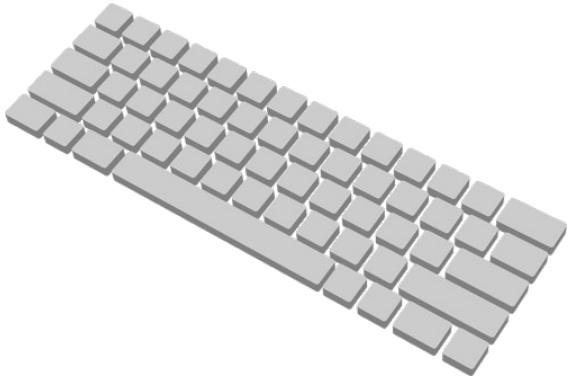
```
#include <iostream> // librairie pour les entrées/sorties (affichage ici)

int main()
{
    // Ceci est notre premier programme
    std::cout << "Hello World" << "\n"; // pour afficher

    std::cin.get(); // pour lire un caractère au clavier : mise en attente!
    return 0;
}
```

Pourquoi std::cout ?
L'espace de noms *std*

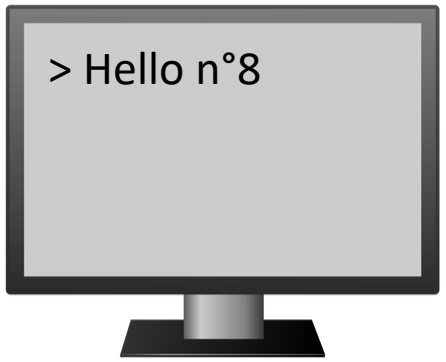
```
#include <iostream>


int main()
{
	// Ceci est notre premier programme
	std::cout << "Hello World" << "\n";
	std::cin.get();
	return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
	// Ceci est notre premier programme
	cout << "Hello World" << "\n";
	cin.get();
	return 0;
}
```

cin, cout : des flux (*stream*)



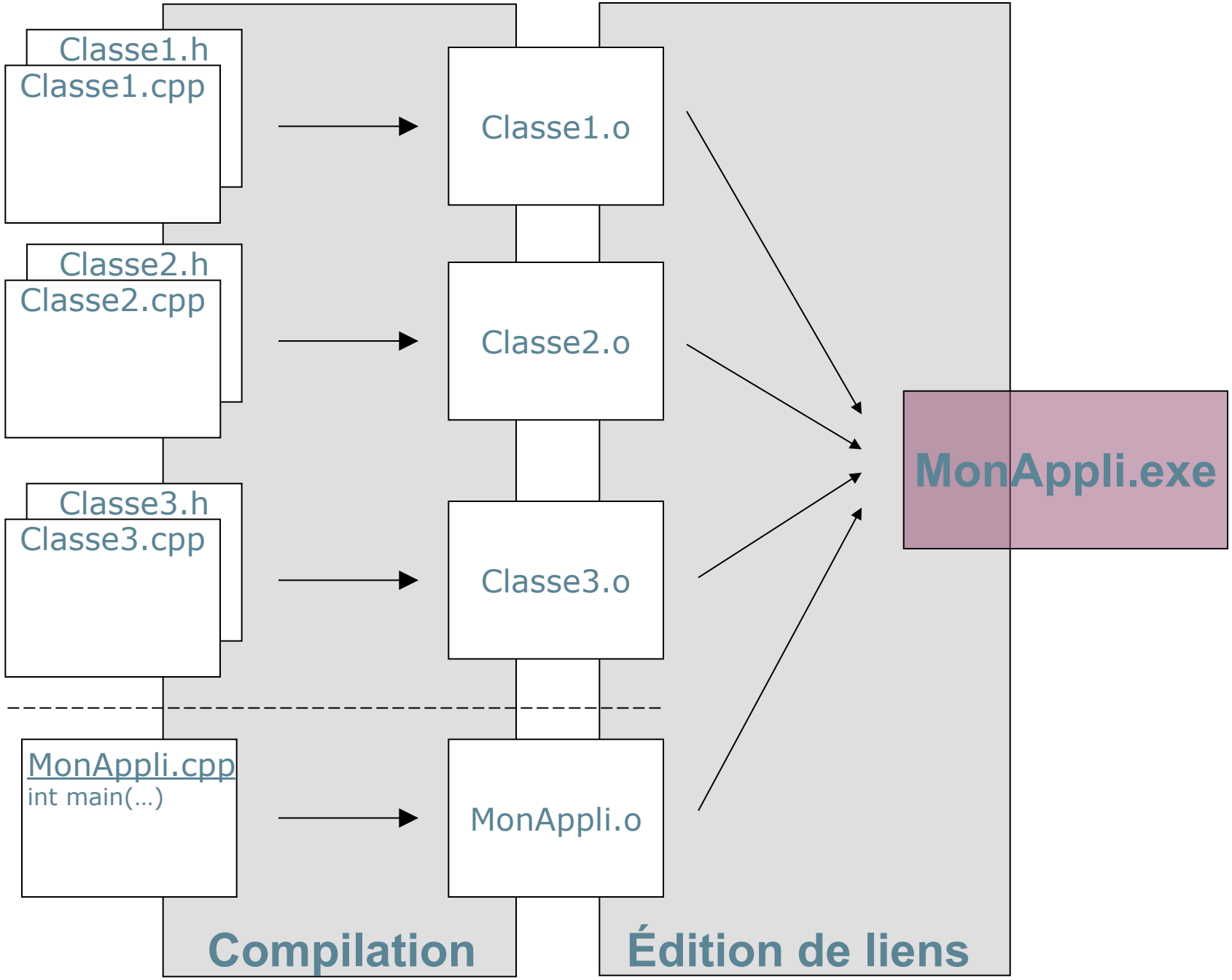
char c; cin >> c;



int a =8; cout << ''Hello n°'' << a;

Comment écrire un programme en C++

- Écrire le code source (.cpp)
- Compiler le code source (code objet .o)
  - Traduire le code source en langage machine
  - Langage machine = propre à chaque système
  - Autant de compilateurs que de systèmes
- Lier le code objet avec d'autres codes objets
  - Autres programmes
  - Librairies
- Génère un exécutable
  - .exe sous Windows
  - Sans extension sous UNIX


Classe1.h
Classe1.cpp
Classe2.h
Classe2.cpp
Classe3.h
Classe3.cpp
MonAppli.cpp
int main(…)
Classe1.o
Classe2.o
Classe3.o
MonAppli.o
MonAppli.exe
Compilation
Édition de liens

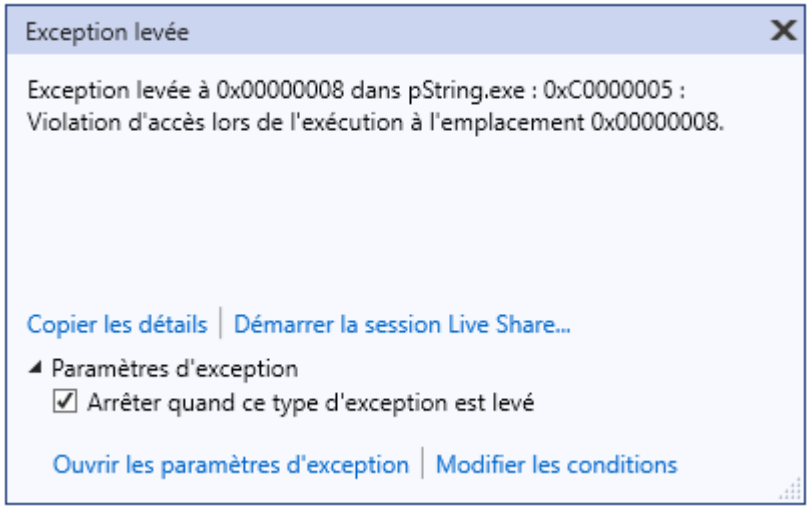
```
class Mesure {
public:
        int data[12];
};

int main()
{

    Mesure m;
    m.data[4] = 9;
    for (int i = 0; i < 12; i++) {

        cout << m.data[i] << endl;
    }
}
```

```
-858993460
-858993460
-858993460
-858993460
9
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
```

```
class Mesure {
public:
        int data[12]={0};
};

int main()
{

    Mesure m;
    m.data[4] = 9;
    for (int i = 0; i < 12; i++) {

        cout << m.data[i] << endl;
    }
}
```

```
0
0
0
0
9
0
0
0
0
0
0
0
0
```

```
m.data[15] = 8;
```


Exception levée
Exception levée à 0x00000008 dans pString.exe : 0xC0000005 :
Violation d'accès lors de l'exécution à l'emplacement 0x00000008.
Copier les détails
Démarrer la session Live Share...
Paramètres d'exception
Arrêter quand ce type d'exception est levé
Ouvrir les paramètres d'exception
Modifier les conditions

Une solution : l'encapsulation

```
class Mesure {
private :
        int data[12] = { 0 };
public :
        void setData (int indice, int donnee) {
                if (indice >= 0 && indice < 12)
                        data[indice] = donnee;
        }
        int getData (int indice) {
                if (indice >= 0 && indice < 12)
                        return data[indice];
                else return -1; }
};
```

```
int main()
{
   Mesure m;
   //m.data[4] = 9;  // Ne fonctionne plus

   m.setData(15, 8); // KO à la compilation : indice!
   m.setData(4, 9); // OK fonctionne
   for (int i = 0; i < 12; i++) {
      //cout << m.data[i] << endl; // Ne fonctionne plus
      cout << m.getData(i) << endl; // OK fonctionne
   }
}
```

Et si on copiait !

```
Mesure m;
m.setData(4, 9);


Mesure m2 = m;
m2.setData(8,12);
```

```
0
0
0
0
9
0
0
0
12
0
0
0
```

Affichage de m2

```
0
0
0
0
9
0
0
0
0
0
0
0
```

Affichage de m

Quelques améliorations

```
class Mesure {
private:
    string name; // un champ nom (#include <string>)
    int data[12] = { 0 };
public :
    Mesure(string n) { name = n; }  // Constructeur
    void setData(int indice, int donnee) {
        if (indice >= 0 && indice < 12)
           data[indice] = donnee; }
    int getData(int indice) {
        if (indice >= 0 && indice < 12)
           return data[indice];
        else return -1; }
    void display() {
        cout << name << endl;
        for (int i = 0; i < 12; i++) {
          cout << data[i] << endl; }
    }
};
```

```
int main()
{
    Mesure m("Thermometre");
    m.setData(4, 9);

    Mesure m2 = m;
    m2.setData(8,12);

    m2.display();
    m.display();
}
```

Constructeur par défaut

```
Mesure m("Thermometre"); // OK
Mesure m; //KO : ne marche plus
```

Avant il n'y avait pas de constructeur, `Mesure m` fonctionnait!
Si vous écrivez votre propre constructeur, vous êtes obligés de l'utiliser!!

Sans constructeur, le système vous en avait fourni un (sans paramètre)
Avec votre constructeur, le système pense que vous n'avez plus besoin de celui par défaut

Vous pouvez le réécrire (le constructeur par défaut, un constructeur sans paramètre)

- Si vous en avez besoin
- Pour créer des tableaux d'objets (car appel au constructeur par défaut)

```
Constructeur par défaut : Mesure() { name = "defaut"; }
```

Construction dynamique/statique

```
Mesure m("Thermometre");
Mesure m2;
```

Se font avec des allocations statiques

Créés dans la pile d'exécution de la fonction
L'objet est détruit (nettoyé) à la fin de la fonction

```
Mesure* pm = new Mesure("Hygromètre");
Mesure* pm2 = new Mesure();
```

Se font avec des allocations dynamiques

Créés dans la mémoire disponible de l'ordinateur
L'objet n'est pas détruit 'naturellement'
C'est à vous de la faire

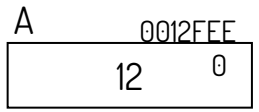
Ce qui est détruit dans la fonction, c'est le pointeur qui pointe vers cette mémoire allouée dynamiquement
C'est à vous de détruire cette mémoire sinon elle sera perdue (fuite mémoire)

```
delete pm; delete pm2;
```

Valeur, adresse, référence : Faites votre choix

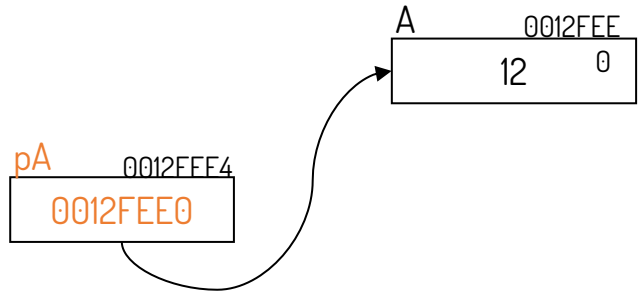
int A = 12;

A est une variable dont le contenu est 12, son adresse dans la mémoire est &A. Ici, cette adresse vaut 0012FEE0;



int* pA=&A;

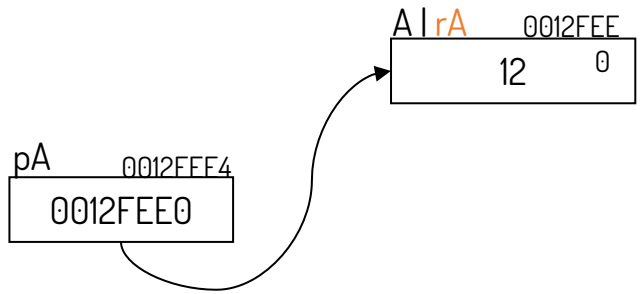
pA est une variable de type pointeur.
Elle a donc une adresse qui lui est propre (ici, &pA, l'adresse de pA vaut 0012FEE4).
Son contenu vaut l'adresse de A
cout << pA affiche 0012FEE0.
Pour afficher le valeur de ce que pointe pA (le contenu de A), il faut déréférencer le pointeur
cout << *pA affiche 12



int& rA = A;

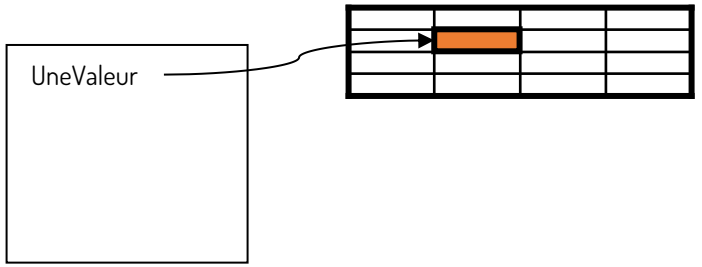
rA est une référence sur la variable A, son adresse est la même que A (&rA = 0012FEE0).
Par conséquent, son contenu est le même que celui de A :

cout << ra affiche 12



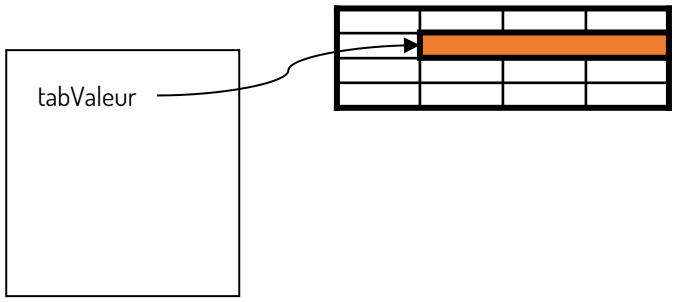
Allocation & tableaux

```
int *uneValeur = new int;
Ville *Lyon = new Ville;
Ville *France = new
Ville[36000];
```



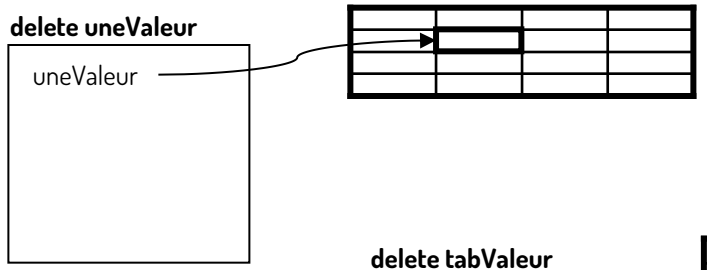
Pourquoi avec les tableaux, on retourne également int * ?

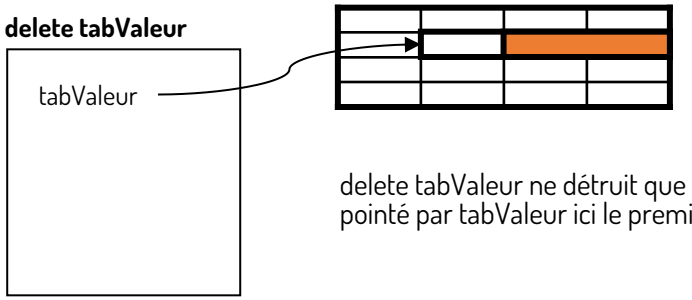
Ex : int *tabValeur = new int[3];



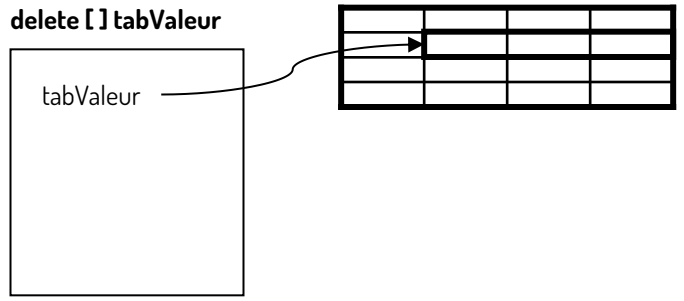
Desallocation

```
delete Lyon;
delete [ ]
France;
```





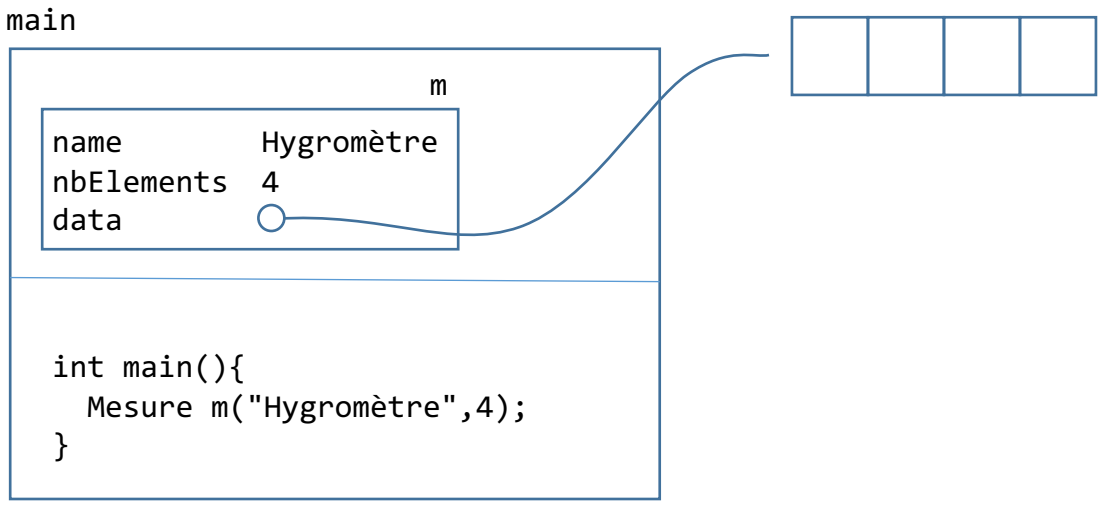
delete tabValeur ne détruit que l'élément pointé par tabValeur ici le premier.



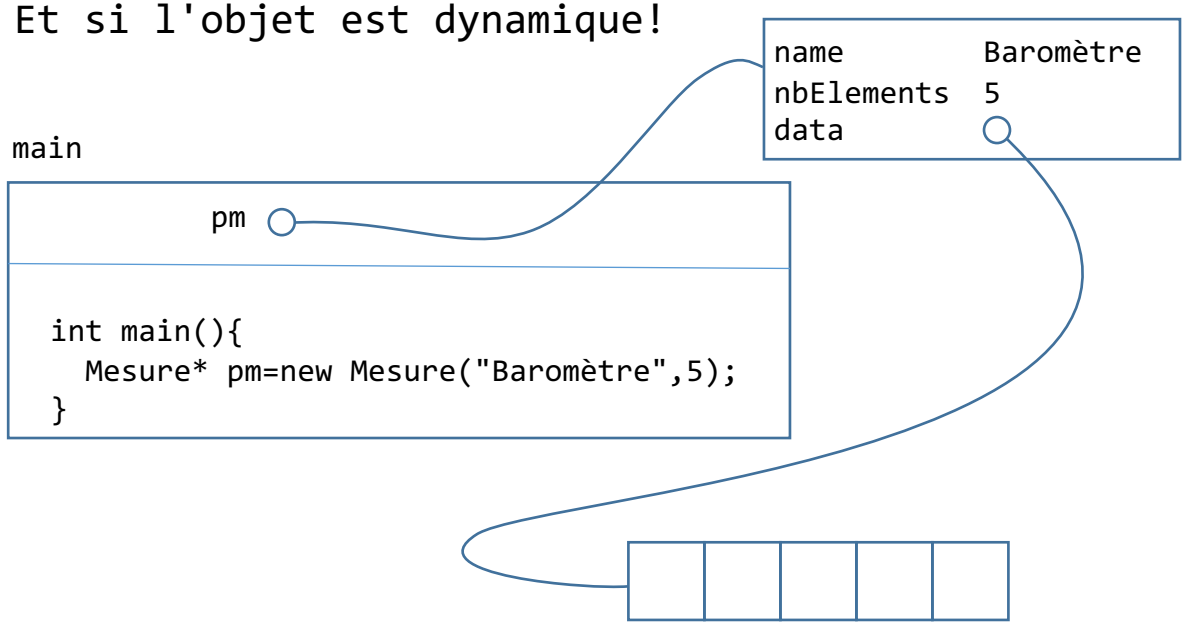
Et si les données de notre classe étaient dynamiques

```
class Mesure {
private:
    string name;
    int nbElements;
    int *data;
```

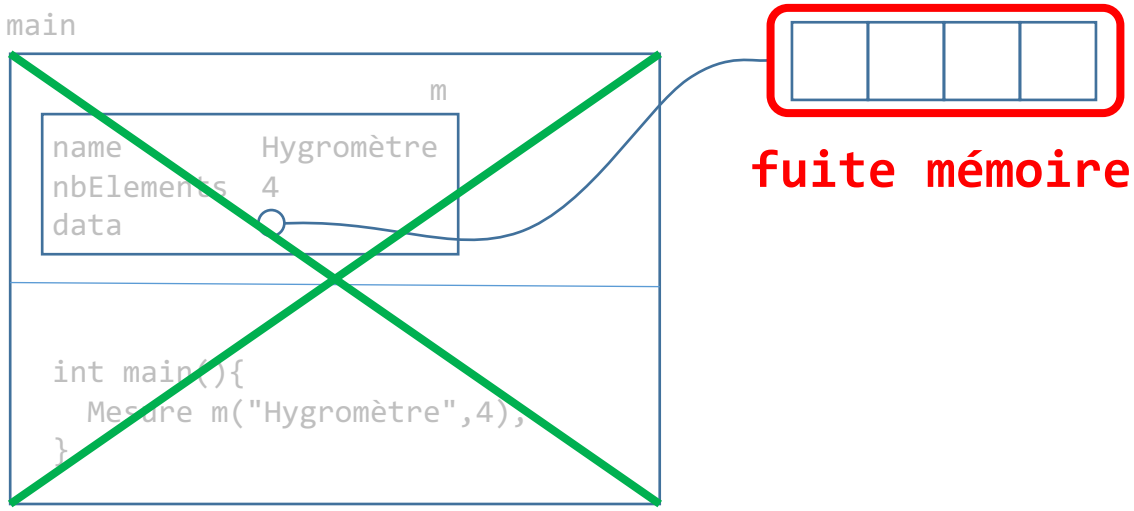
```
class Mesure {
    private:
        string name;
        int nbElements;
        int *data;
    public :
        Mesure() { name = "defaut"; }
        Mesure(string n, int nbE) {
                name = n; nbElements = nbE; data = new int[nbE]; for (int i = 0; i < nbE; i++) { data[i] = 0; }
        }
        void setData(int indice, int donnee) {
                if (indice >= 0 && indice < nbElements) data[indice] = donnee; }
        int getData(int indice) {
                if (indice >= 0 && indice < nbElements)
                        return data[indice];
                else return -1; }
        void display() {
             cout << name << endl;
             for (int i = 0; i < nbElements; i++) {
                  cout << data[i] << endl;
             }
        }
};
```



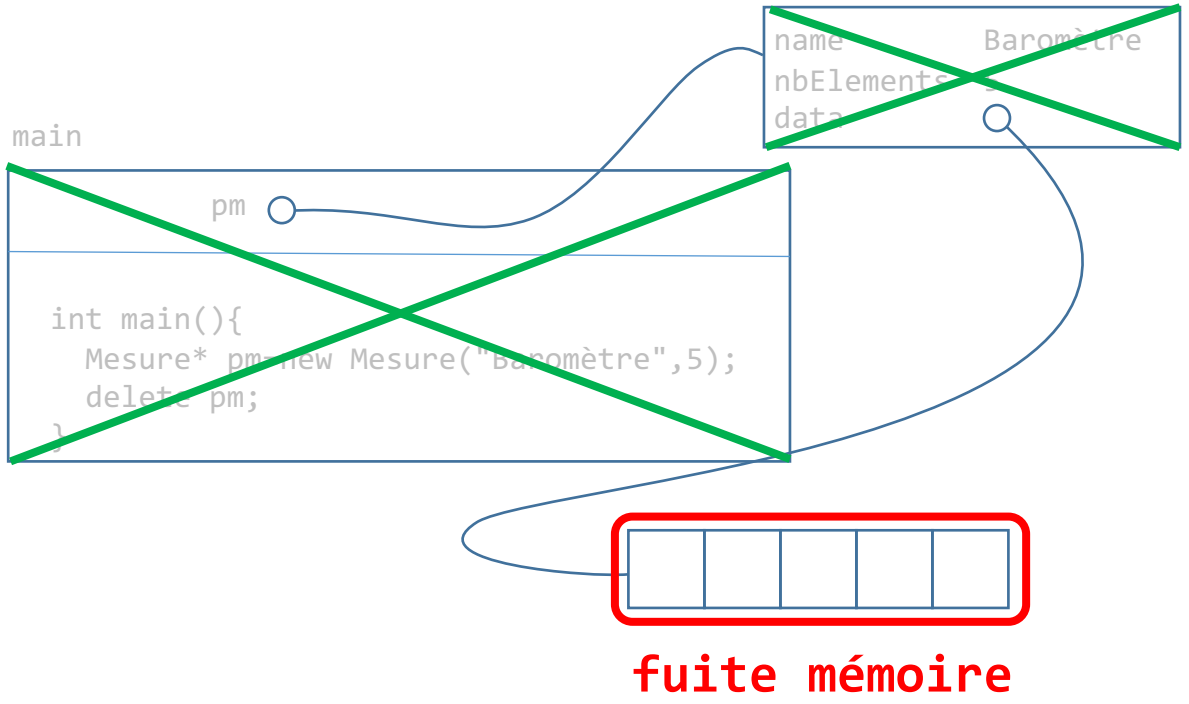
Et si l'objet est dynamique!



Et le nettoyage ??



fuite mémoire



fuite mémoire

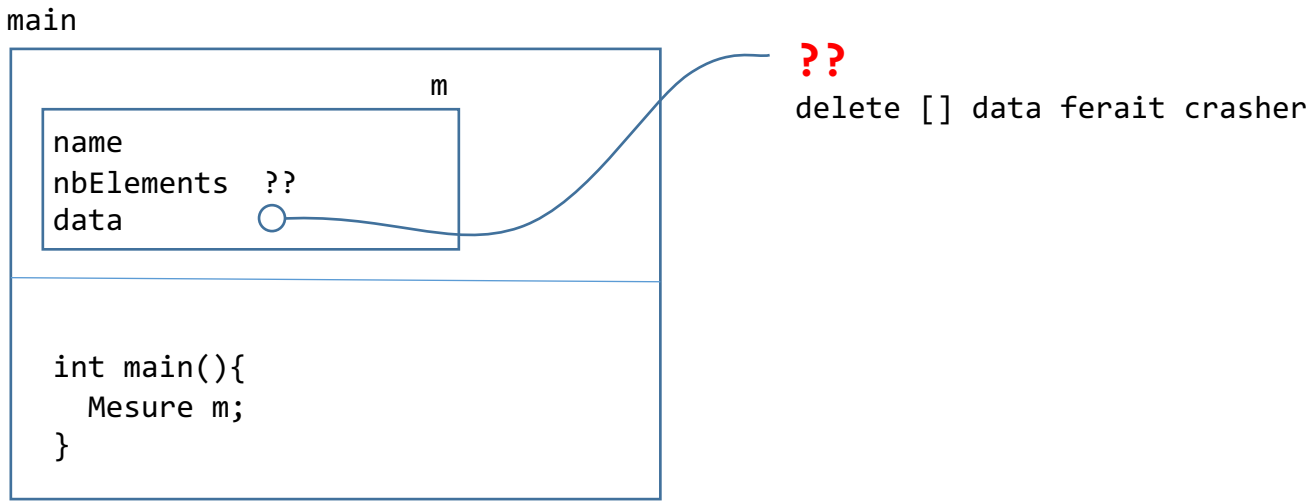
Un destructeur
pour assurer le nettoyage

Nettoyage de la partie dynamique

```
~Mesure() { if (data != nullptr) delete [] data; }
```

~NomClasse() pour définir un destructeur

Pourquoi ce test ?




```
class Mesure {
private:
      string name;
      int nbElements;
      int *data = nullptr;
public :
       ~Mesure() { if (data != nullptr) delete[] data; }
```

Revenons un peu en arrière

```
int main()
{
    Mesure m("Thermometre");
    m.setData(4, 9);

    Mesure m2 = m;
    m2.setData(8,12);

    m2.display();
    m.display();
}
```

on avait obtenu çà !

```
0
0
0
0
9
0
0
0
12
0
0
0
```

Maintenant

```
0   0
0   0
0   0
9   0
0   9
0   0
0   0
12  0
0   12
0   0
0   0
    0
```

Affichage de m2

```
0   0
0   0
0   0
9   0
0   9
0   0
0   0
12  0
0   12
0   0
0   0
    0
```

Affichage de m2

```
0   0
0   0
0   0
9   0
0   9
0   0
0   0
12  0
0   12
0   0
0   0
    0
```

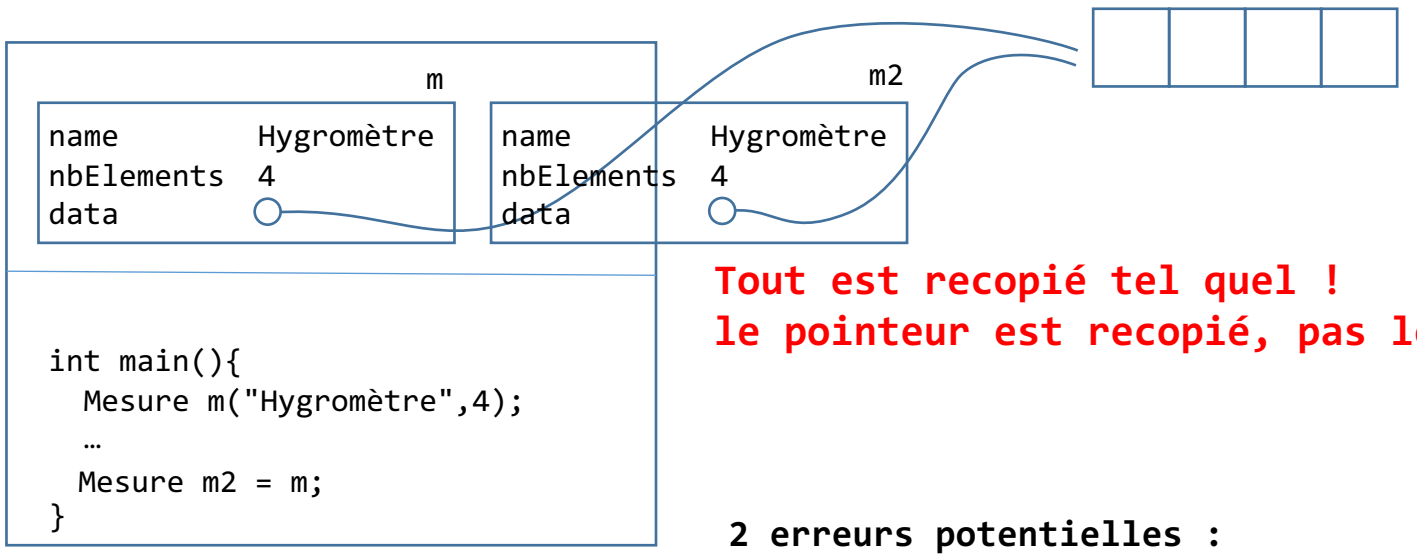
Affichage de m

```
0   0
0   0
0   0
9   0
0   9
0   0
0   0
12  0
0   12
0   0
0   0
    0
```

Affichage de m

Une copie par défaut trop 'simpliste'

```
Mesure m2 = m;
```



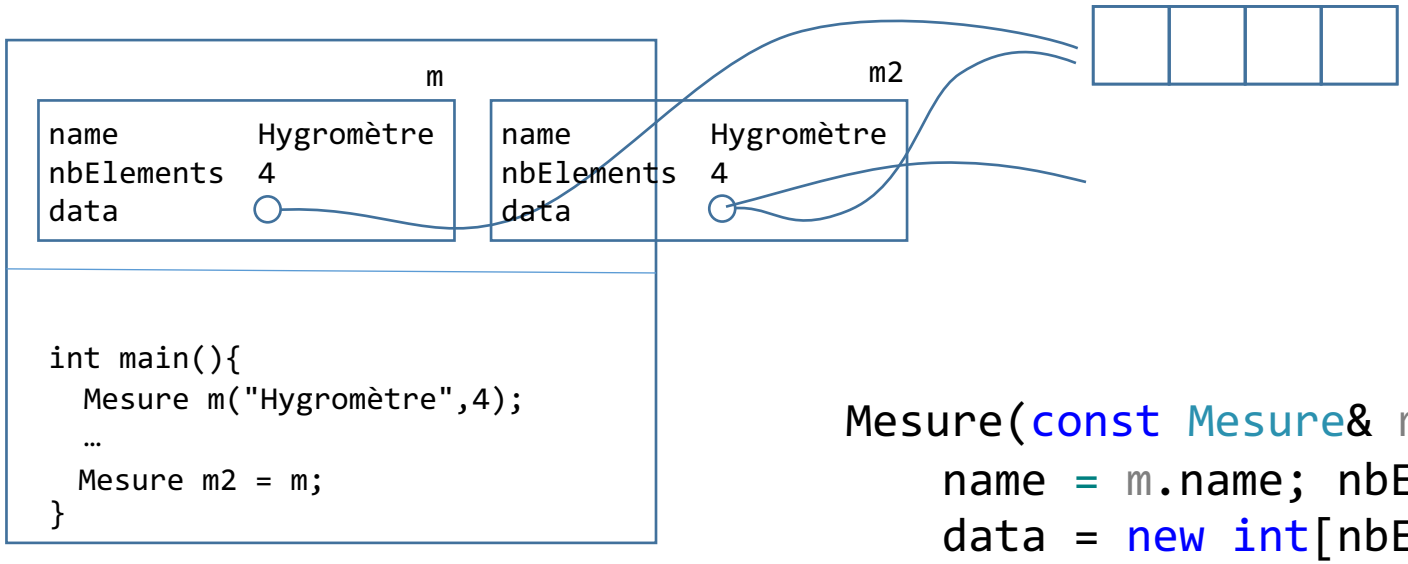
Tout est recopié tel quel !
le pointeur est recopié, pas les données !

2 erreurs potentielles :

- Le tableau des données est partagé entre l'objet et sa copie
- À la destruction : m et m2 vont être détruits à la fin de la fonction main sauf que l'appel au destructeur va essayer de détruire deux fois le tableau de données

le constructeur par copie, une copie plus efficace

```
Mesure m2 = m;
```




```
Mesure(const Mesure& m) {
    name = m.name; nbElements = m.nbElements;
    data = new int[nbElements];
    for (int i = 0; i < nbElements; i++) {
        data[i] = m.data[i];
    }
}
```