1. Etudier le programme initial.pdf
2. Prédire ce qui va s'afficher - 5 minutes

   Le résultat :

```
t1 : 1, 2, 3, 4, 5
t2 : 1, 2, 3, 4, 5
t2 : 1, 42, 3, 4, 5
t1 : 1, 42, 3, 4, 5
```

   Surpris(e) ?

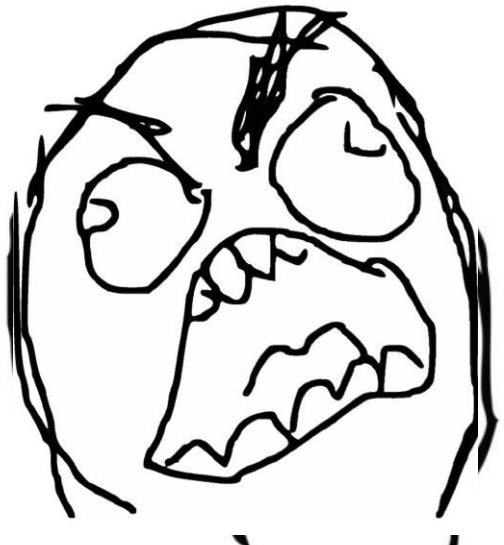
3. Ajoutons un mouchard

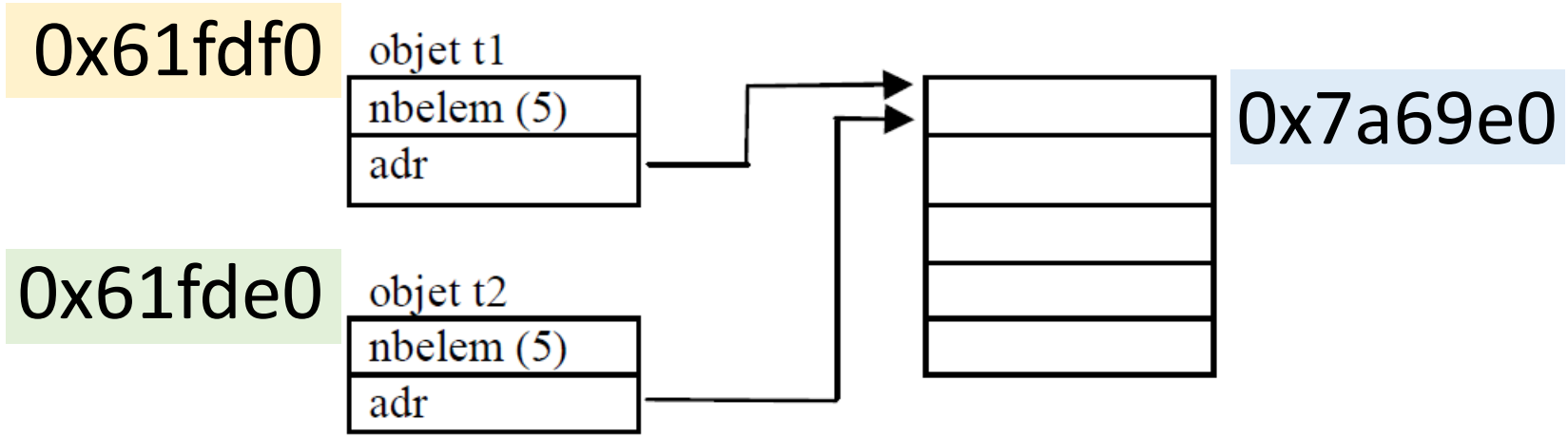
```cpp
#include <iostream>

using namespace std;

class tab {
    private :
        int nbElem; // Nombre d'éléments dans le tableau
        int * adr;  // Adresse du 1er élément du tableau
    public :
        tab() {
            nbElem = 0;
            cout << "C - @obj : " << this << ", adr : " << adr << endl;
        }
        tab(int n) {
            nbElem = n;
            adr = new int[nbElem];
            cout << "C - @obj : " << this << ", adr : " << adr << endl;
        }
        ~tab() {
            cout << "D - @obj : " << this << ", adr : " << adr << endl;
            delete [] adr;
        }
        void set(int i, int value) {
            if (i < nbElem) {
                adr[i] = value;
            }
        }
        int get(int i) {
            int result = 0;
            if (i < nbElem) {
                result = adr[i];
            }
            return result;
        }
        void display() {
            for (int i = 0; i < nbElem; i++) {
                cout << adr[i] << (i < nbElem - 1 ? ", " : "");
            }
            cout << endl;
        }

};

int main() {
    tab t1(5);
    for (int i = 0; i < 5; i++) {
        t1.set(i, i + 1);
    }

    cout << "t1 : ";
    t1.display();

    tab t2 = t1;

    cout << "t2 : ";
    t2.display();

    cout << "t2 : ";
    t2.set(1, 42);
    t2.display();

    cout << "t1 : ";
    t1.display();

    return 0;
}
```

Le résultat :

```
C - @obj : 0x61fdf0, adr : 0x7a69e0
t1 : 1, 2, 3, 4, 5
t2 : 1, 2, 3, 4, 5
t2 : 1, 42, 3, 4, 5
t1 : 1, 42, 3, 4, 5
D - @obj : 0x61fde0, adr : 0x7a69e0
D - @obj : 0x61fdf0, adr : 0x7a69e0
```

?




0x61fdf0
objet t1
nbelem (5)
adr
0x61fde0
objet t2
nbelem (5)
adr
0x7a69e0

Conclusion

- Le mécanisme standard de recopie d’objet, recopie les valeurs de t1 dans t2, même les adresses !

- Il n’utilise pas les constructeurs « normaux ».

Comment l'éviter ?

- Pour éviter cela, il faut mettre en place un **constructeur de recopie** :

```
NomClasse (NomClasse &);
```

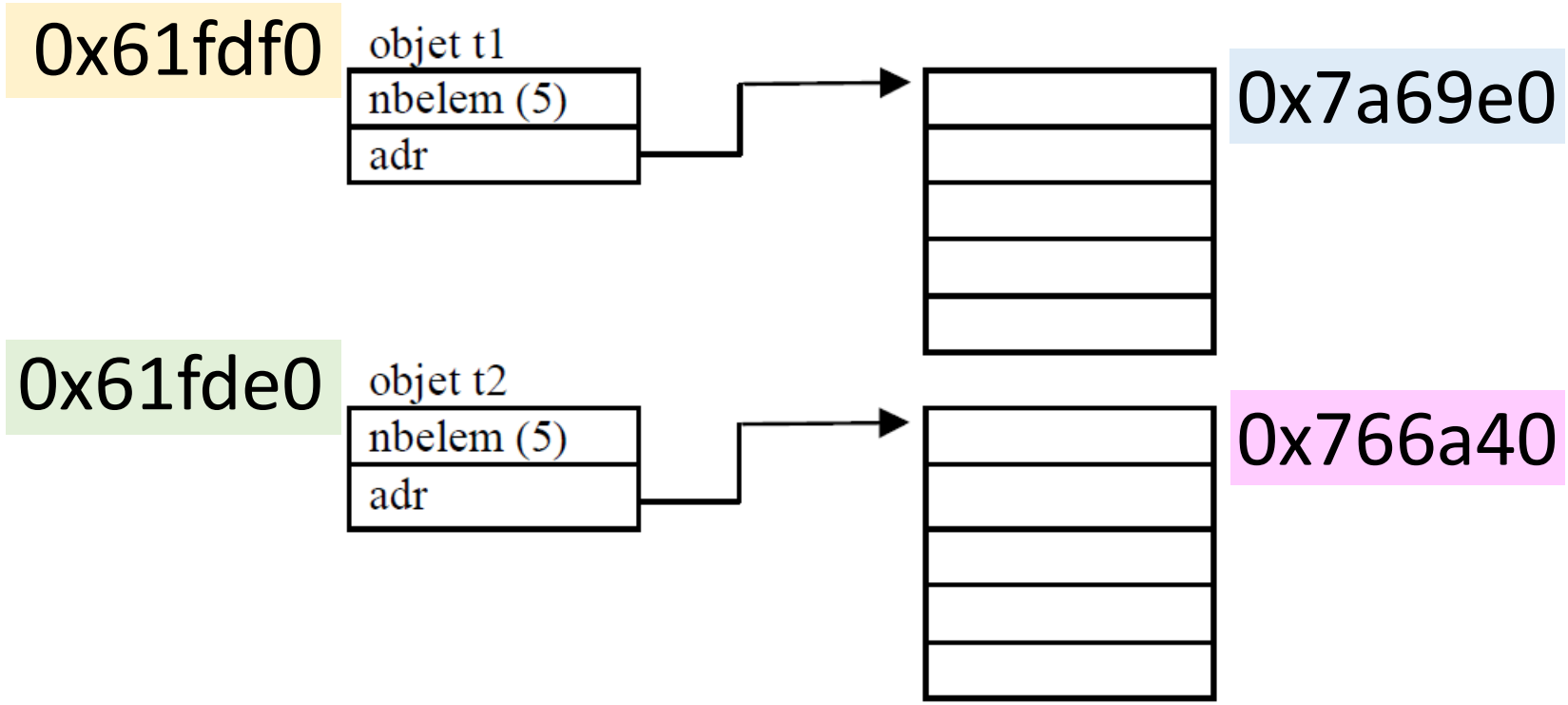
pour y implémenter le fonctionnement attendu.

- Il est appelé :
  - à l'**initialisation** d'un objet par un objet de même type ;
  - lors d'un **passage par valeur** d'un objet à une fonction.

```
// Constructeur par recopie pour le programme initial
tab (tab & t) {
    // Recopie du nombre d’éléments
    nbElem = t.nbElem;
    // Création d'un nouveau tableau
    adr = new int [nbElem];
    // Recopie des valeurs du tableau à recopier
    for (int i = 0; i <  nbElem; i++) {
        adr[i] = t.adr[i];
    }
}
```

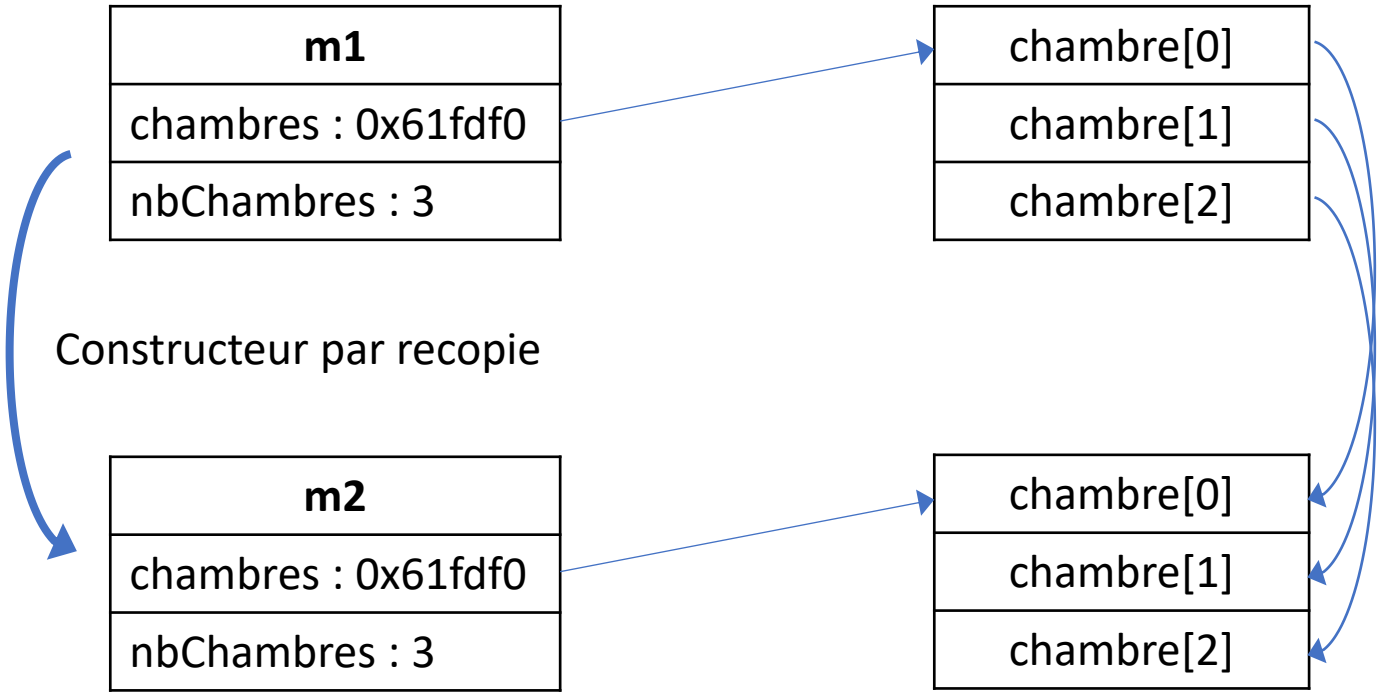
Le résultat :

```
C - @obj : 0x61fdf0, adr : 0x7669e0
t1 : 1, 2, 3, 4, 5
R - @obj : 0x61fde0, adr : 0x766a40
t2 : 1, 2, 3, 4, 5
t2 : 1, 42, 3, 4, 5
t1 : 1, 2, 3, 4, 5
D - @obj : 0x61fde0, adr : 0x766a40
D - @obj : 0x61fdf0, adr : 0x7669e0
```


0x61fdf0
objet t1
nbelem (5)
adr
0x7a69e0
0x61fde0
objet t2
nbelem (5)
adr
0x766a40

```
int main() {
    tab t1(5);
    tab t2(3);
    t2 = t1;    // ???
    return 0;
}
```

La suite au prochain épisode...


m1
chambres : 0x61fdf0
nbChambres : 3
chambre[0]
chambre[1]
chambre[2]
Constructeur par recopie
m2
chambres : 0x61fdf0
nbChambres : 3
chambre[0]
chambre[1]
chambre[2]