

P00 – Langage C++

Les classes et les objets (parties 2 et 3)

1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

2019 - 2020

plan

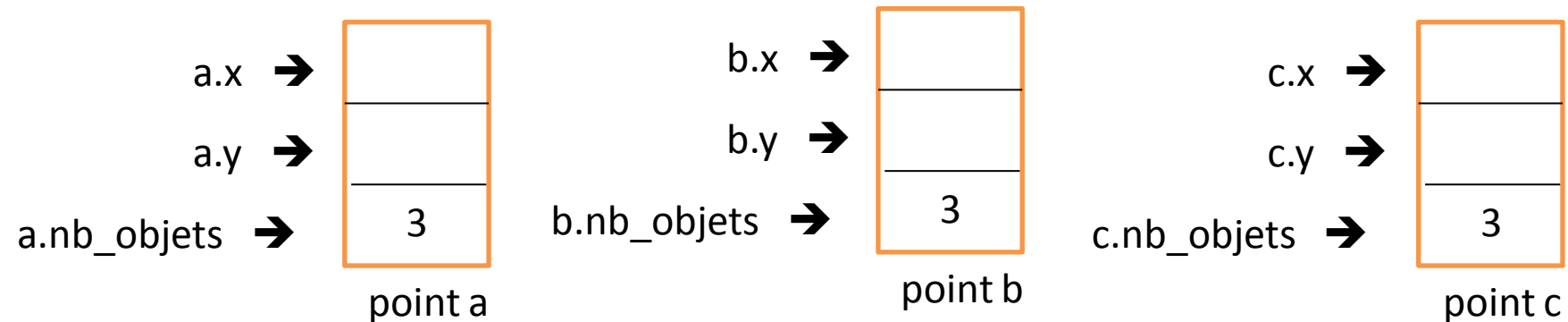
1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Surdéfinition des fonctions membres
6. Arguments par défaut des fonctions membres
7. Les membres données et fonctions statiques
8. Protection contre les inclusions multiples
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. Objet retourné par une fonction
11. Autoréférence: le mot clé this
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

Le qualificatif *static* pour un membre donnée

- lorsque dans un même programme, on crée différents objets d'une même classe, chaque objet possède ses propres membres données.

```
class point
{ int x;
  int y;
  int nb_objets;
// méthodes
};
```

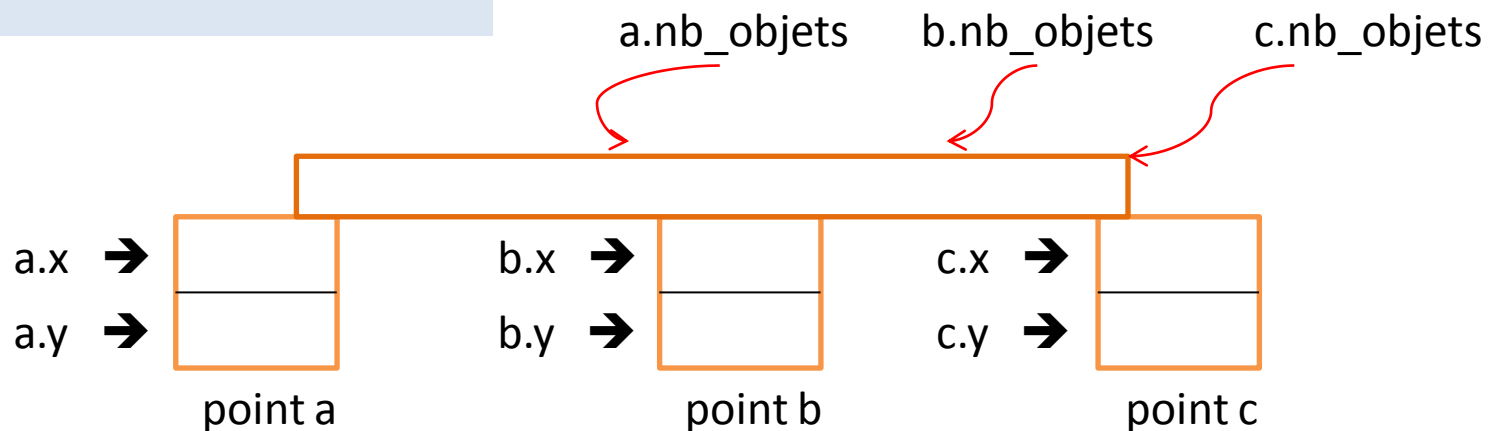
nb_objets est un champ qui contient le nombre total d'objets créés (instanciés) à partir de la classe point



Le qualificatif *static* pour un membre donnée

- Pour permettre à plusieurs objets de **partager** des données, il faut déclarer avec le qualificatif *static* les membres données qu'on souhaite voir exister en **un seul exemplaire** pour tous les objets de la classe.

```
class point
{ int x,y;
  static int nb_objets;
};
```



Initialisation des membres données statiques

- Par leur nature même, les membres données statiques n'existent qu'en **un seul exemplaire**, indépendamment des objets de la classe (**même si aucun objet de la classe n'a encore été créé**).
- leur initialisation **ne peut plus être faite par le constructeur** de la classe, ni dans la classe

```
class point
{
    int x,y;
    static int nb_objets=0; //erreur
    .....};
```

- Un membre statique doit donc être initialisé **explicitement** (dans le fichier .cpp) en utilisant l'opérateur de résolution de portée (:) pour spécifier sa classe.

```
int point::nb_objets=0;
```

Les fonctions membres statiques

- C++ permet de définir des membres **données statiques**.
- Ceux-ci existent **en un seul exemplaire** (pour une classe donnée), **indépendamment** des objets de leur classe.
- D'une manière analogue, **les fonctions membres statiques** d'une classe ont un rôle totalement **indépendant** d'un quelconque objet;
- Elles **agissent sur des membres données statiques**.
- Elles peuvent être appelée de 2 manières:
 - `nomObjet.nomFonctionStatique(..);`
 - `nomClasse::nomFonctionStatique(..);`

- Une fonction statique est une fonction membre qui ne peut accéder qu'aux membres statiques.
- Une telle méthode n'a pas de membre this et peut être appelée indépendamment de tout objet

(Global Scope)

```
class point
{
    int x;
    int y;
    static int nb_objets;
public:
    point(int =99,int =88);
    void deplacer(int,int);
    void afficher(string = "");
    //setteurs et getteurs..
    static void afficher_nb_objets();
    ~point();
};
```

point

~point()

```
#include "point.h"
int point::nb_objets=0;

point::point(int abs,int ord)
{
    cout<<"\n +++ construc " <<this<<endl;
    x=abs;
    y=ord;
    nb_objets++;
}

void point::afficher_nb_objets()
{
    cout<<"\n le nombre total d'objets crees " <<nb_objets<<endl;
}
```


Une fonction membre statique
**peut être appelée même si
aucun objet de la classe n'a été
créé.**

```
point.cpp  point.h  main.cpp x
(Global Scope)
#include "point.h"
void main()
{
    point a(2,3);
    point b(5,6);
    point c(7,8);
    cout<<"\n-----"<<endl;
    a.afficher_nb_objets();
    cout<<"\n-----"<<endl;
    point::afficher_nb_objets();
    cout<<endl;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingC2014\D...
+++ construc  004FFEAC
+++ construc  004FFE9C
+++ construc  004FFE8C

-----

le nombre total d'objets crees 3

-----

le nombre total d'objets crees 3
Appuyez sur une touche pour continuer...
```

plan

1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Surdéfinition des fonctions membres
6. Arguments par défaut des fonctions membres
7. Les membres données et fonctions statiques
8. **Protection contre les inclusions multiples**
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. Objet retourné par une fonction
11. Autoréférence: le mot clé this
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

Protection contre les inclusions multiples

- Nous verrons qu'il existe différentes circonstances pouvant amener l'utilisateur d'une classe à inclure plusieurs fois un fichier en-tête lors de la compilation d'un même fichier
- Ce sera notamment le cas dans les situations **d'objets membres** et de **classes dérivées**.

```
#include "point.h"
class cercle
{ point centre;
  float rayon;
  .....
};
```

```
#include "point.h"
#include "cercle.h"
void main()
{ .....
}
```

- **Résultat:** erreurs de compilation, liées à la redéfinition de la classe concernée.

```
point.h x main.cpp point.cpp
point
#pragma once
class point
{
    int x,y;
public:
    point(int,int);
    void deplacer(int,int);
    void afficher(char * = "");
    ~point();
};
```

```
point.h x main.cpp point.cpp
point
#ifndef POINT_H
#define POINT_H
class point
{
    int x,y;
public:
    point(int,int);
    void deplacer(int,int);
    void afficher(char * = "");
    ~point();
};
#endif
```

euch

plan

1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Surdéfinition des fonctions membres
6. Arguments par défaut des fonctions membres
7. Les membres données et fonctions statiques
8. Protection contre les inclusions multiples
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. Objet retourné par une fonction
11. Autoréférence: le mot clé this
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

Transmission d'un objet en argument d'une fonction

- Une fonction membre peut recevoir en argument, outre l'objet l'ayant appelé (transmis implicitement) un ou plusieurs objets de type classe.
- Nous nous limiterons au cas d'objets de **même type** que la classe dont la fonction est membre;
- Les autres situations, correspondant à une violation du principe d'encapsulation, seront examinées plus tard, (chapitre: les **fonctions amies**)

```
point.cpp  point.h x main.cpp
(Global Scope)
class point
{
private:
    int x;
    int y;
public:
    point(int =99,int =88);
    void deplacer(int,int);
    void afficher(string = "");

    bool coincide (point); // ==> passage par valeur

    bool coincide (point*); // ==> passage par adresse

    bool coincide (point &); // ==> passage par référence
    ~point();
};
```

*coincide vérifie si deux points
coincident: mêmes x et y*

*L'argument de la fonction
coincide est de type point (même
type que la classe)*

© 2011-2012 Vincent Van der Elst

Passage par valeur

```
point.cpp* x point.h* main.cpp
(Global Scope)
bool point::coincide(point pt)
{
    if(x==pt.x && y==pt.y) return true;
    else return false;
}
```

```
point.cpp point.h main.cpp x
(Global Scope)
#include "etudiant.h"
#include "point.h"
void main()
{
    point a(2,3);
    cout<<"\n -----"<
    point b(5,6);
    cout<<"\n -----"<
    bool res=a.coincide(b);

    if(res==1) cout<<"\n coincident "<<endl;
    else cout<<"\n ne coincident pas"<<endl;
    cout<<endl;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\...
+++constructeur 0035FB34
-----
+++constructeur 0035FB24
-----
appel du destructeur 0035FA3C
ne coincident pas
Appuyez sur une touche pour continuer...
```



```
point.cpp* point.h* main.cpp
(Global Scope)
class point
{
    int x;
    int y;
public:
    point(int =99,int =88);
    void deplacer(int,int);
    void afficher(string = "");
    //setteurs et getteurs..
    ~point();
    // passage par valeur:
    bool coincide (point);
    // passage par adresse:
    bool coincide (const point*);
    // passage par référence:
    bool coincide (const point&);
};
```

```
point.cpp* point.h* main.cpp
point
bool point::coincide(point pt)
{ // passage par valeur
    if(x==pt.x && y==pt.y) return 1;
    else return 0;
}
```

```
point.cpp* point.h* main.cpp
(Global Scope)
bool point::coincide(const point *pt)
{ // passage par adresse
    if(x==pt->x && y==pt->y) return 1;
    else return 0;
}
```

```
point.cpp* point.h* main.cpp
(Global Scope)
bool point::coincide(const point &pt)
{ // passage par référence
    if(x==pt.x && y==pt.y) return 1;
    else return 0;
}
```

Appel de coincide

```
void main()
{
    point a(2,3);
    point b(5,6);
    cout<<"\n-----"<<endl;
    bool res=a.coincide(b);

    // avec cet appel, il y a ambiguïté:
    // appel de
    // bool coincide (point);
    // ou bien
    // bool coincide (const point&);
    if(res==1) cout<<"\n coïncident "<<endl;
    else cout<<"\n ne coïncident pas"<<endl;
    system("PAUSE");
}
```

Error: more than one instance of overloaded function "point::coincide" matches the argument list:

```
void main()
{
    point a(2,3);
    point b(5,6);
    cout<<"\n-----"<<endl;
    bool res=a.coincide(&b);
    //appel de bool coincide (const point*);

    if(res==1) cout<<"\n coïncident "<<endl;
    else cout<<"\n ne coïncident pas"<<endl;
    system("PAUSE");
}
```

```
point.cpp* x point.h main.cpp
point
bool point::coincide(point pt)
{
    cout<<this<<endl;
    cout<<"\n l'adresse de pt "<<&pt<<endl;
    if(x==pt.x && y==pt.y) return true;
    else return false;
}
```

```
point.cpp point.h main.cpp x
(Global Scope)
#include"point.h"
void main()
{
    point a(2,3);
    cout<<"\n -----"<<endl;
    point b(5,6);
    cout<<"\n -----"<<endl;
    bool res=a.coincide(b);
    cout<<"\n -----"<<endl;
    if(res==1) cout<<"\n coïncident "<<endl;
    else cout<<"\n ne coïncident pas"<<endl;
    cout<<endl;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\De...
+++constructeur 002FF7FC
-----
+++constructeur 002FF7EC
-----
002FF7FC
l'adresse de pt 002FF704
appel du destructeur 002FF704
-----
ne coïncident pas
Appuyez sur une touche pour continuer.
```

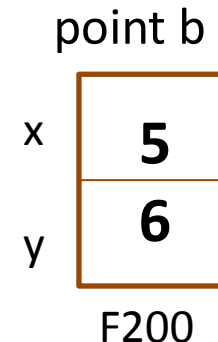
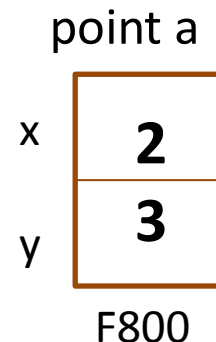
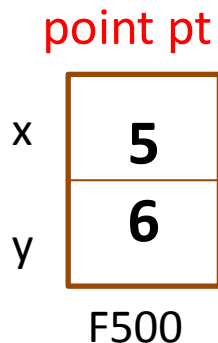
Passage par valeur

// en-tête

```
bool point::coincide (point pt)
```

Appel dans main:

```
bool res;  
point a(2,3);  
point b(5,6);  
res=a.coincide(b);
```



- l'**argument** de la fonction coincide est un **objet** qui s'appelle pt
- pt est un **nouvel** objet qui s'est créé au moment de l'appel de coincide (il a sa propre adresse mémoire: F500).
- pt est **initialisé** par les valeurs de l'objet b (5 et 6).
- il est **détruit** au moment de la sortie de la fonction

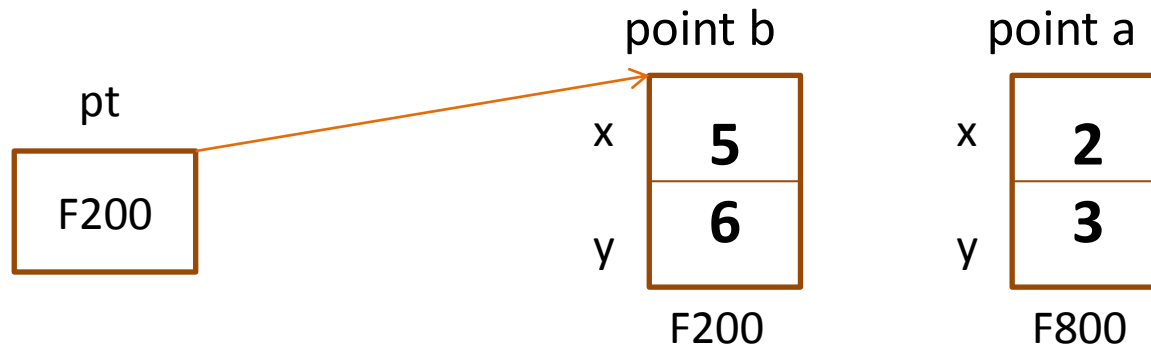
Passage par adresse

// en-tête

```
bool point::coincide (point *pt)
```

Appel dans main:

```
bool res;  
point a(2,3);  
point b(5,6);  
res=a.coincide(&b);
```



pt est un pointeur sur l'objet b

Problème: Dans la fonction `coincide`, `pt` peut modifier les valeurs (`x` et `y`) de l'objet `b` (`pt->x=99; pt->y=88;`)

Solution:

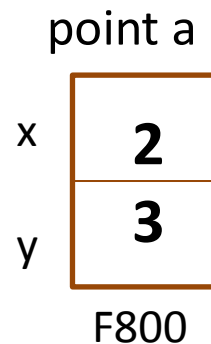
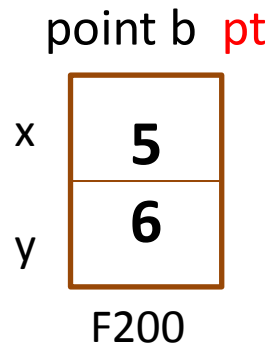
```
bool coincide (const point *); // point.h  
bool point:: coincide (const point *pt) // point.cpp
```

Passage par référence

Appel dans main:
bool res;
point a(2,3);
point b(5,6);
res=a.coincide(b);

// en-tête

bool point::coincide (point &pt)



pt est une référence sur l'objet b

Problème: Dans la fonction coincide, l'objet b peut être modifié par la référence pt (**pt.x=99; pt.y=88;**)

Solution:

bool coincide (**const** point &); // point.h
bool point:: coincide (**const** point &pt) // point.cpp

Passage par valeur, adresse et référence

Appel	En-tête <i>// ajouter point::</i>	Schéma
<i>// passage par valeur</i> a.coincide(b);	bool coincide (point pt) Création de pt à partir de b. pt est <i>une copie</i> de b	<div> <div>Objet b</div> <div> <div>5</div> <div>6</div> </div> <div>F800</div> </div> <div> <div>Objet pt</div> <div> <div>5</div> <div>6</div> </div> <div>F500</div> </div>

méthode qui retourne l'objet qui l'a appelée: ***this**

- Inferieur: est une méthode de la classe point qui retourne l'objet ayant l'attribut x, le plus petit

```
point
class point
{
    int x;
    int y;
    static int nbObjets;
public:
    point(int =99,int =88);
    point inferieur(point);
};

// point c= a.inferieur(b);
point point::inferieur(point pt)
{
    if(x<pt.x) return *this;
    return pt;
}

(Global Scope)
void main()
{
    point a(11,22);
    point b(44,55);
    point c=a.inferieur(b);

    // cette fonction retourne
    // soit l'objet a, soit l'objet b
    c.afficher("point inferieur");
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\POOD2...
+++ appel constr 2 arg point +++ 0044F870
+++ appel constr 2 arg point +++ 0044F860
--- appel destr point --- 0044F764
point inferieur
abscisse 11 ordonnee 22 0044F850
Appuyez sur une touche pour continuer...
```


plan

1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Les membres données et fonctions statiques
6. Protection contre les inclusions multiples
7. Surdéfinition des fonctions membres
8. Arguments par défaut des fonctions membres
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. **Objet retourné par une fonction**
11. Autoréférence: le mot clé this
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

```

class point
{
    int x;
    int y;
public:
    point(int =99,int =88);
    void deplacer(int,int);
    void afficher(string = "");
    //setteurs et getteurs..
    ~point();
    bool coincide (const point*);
    bool coincide (const point&);
    point symetrique();
};

```

point.cpp* x point.h* main.cpp*

(Global Scope)

```

point point::symetrique()
{
    point pt;
    pt.x=-x;
    pt.y=-y;
    return pt;
}

```

point.cpp point.h main.cpp* x

(Global Scope)

```

#include"point.h"
void main()
{
    point a(2,3);
    cout<<"\n-----"
    point b=a.symetrique();
    cout<<"\n-----"
    b.afficher("POINT b");
    cout<<endl;
    system("PAUSE");
}

```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingC2014\Debug\i...

```

+++ construc 003AF894
-----
+++ construc 003AF770
--- appel du destructeur 003AF770
-----
POINT b 003AF884 coordonnees -2 -3
Appuyez sur une touche pour continuer...

```

Fonction renvoie un objet

```
point point::symetrique ()  
{  
    point pt;  
    pt.x=-x;    pt.y =-y;  
    return pt;  
}
```

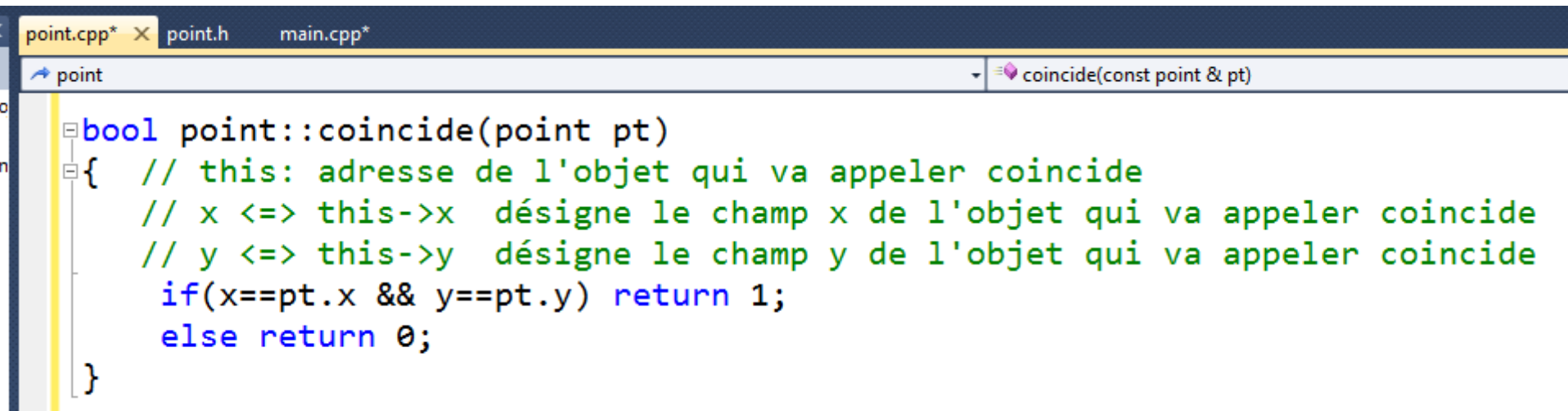
```
point & point :: symetrique () //déconseillé  
point * point :: symetrique () //déconseillé
```

plan

1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Les membres données et fonctions statiques
6. Protection contre les inclusions multiples
7. Surdéfinition des fonctions membres
8. Arguments par défaut des fonctions membres
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. Objet retourné par une fonction
11. **Autoréférence: le mot clé this**
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

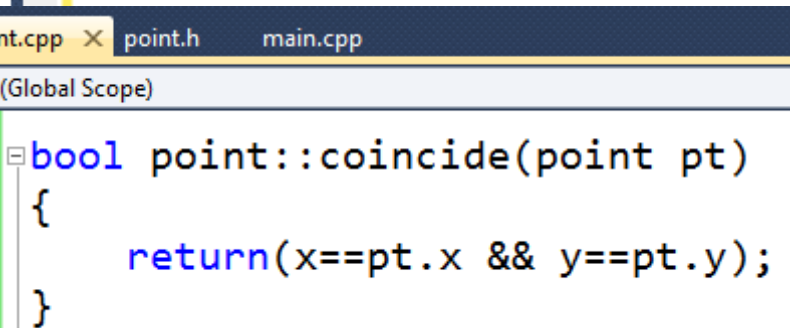
Le mot clé this

- This est utilisable uniquement au sein d'une fonction membre,
- Il désigne **un pointeur** sur l'objet l'ayant appelé.
- Exemple:



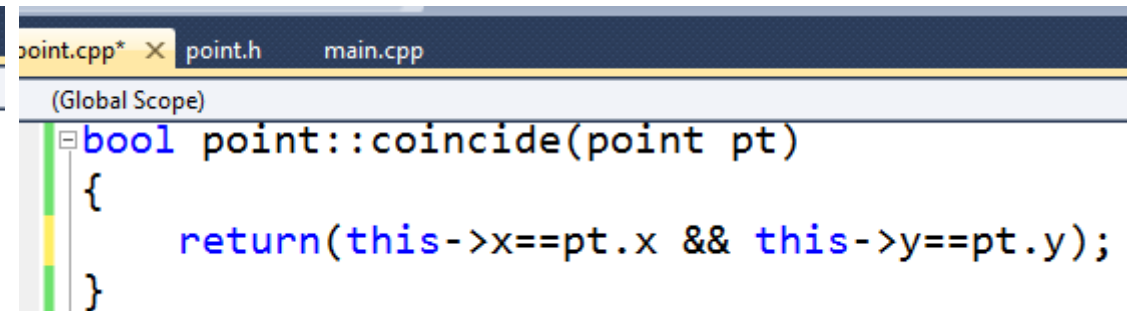
```
point.cpp* x point.h main.cpp*
point coincide(const point & pt)

bool point::coincide(point pt)
{
    // this: adresse de l'objet qui va appeler coincide
    // x <=> this->x désigne le champ x de l'objet qui va appeler coincide
    // y <=> this->y désigne le champ y de l'objet qui va appeler coincide
    if(x==pt.x && y==pt.y) return 1;
    else return 0;
}
```



```
nt.cpp x point.h main.cpp
(Global Scope)

bool point::coincide(point pt)
{
    return(x==pt.x && y==pt.y);
}
```



```
point.cpp* x point.h main.cpp
(Global Scope)

bool point::coincide(point pt)
{
    return(this->x==pt.x && this->y==pt.y);
}
```

Exemple facture

```
facture
#include<iostream>
using namespace std;
#include<string>
class facture
{
    int cf;
    string libelle;
    int nbArticles;
    float*articles; // prix des articles
    float total;

public:
    facture(int =99, string ="", int =2);
    void saisirPrix();
    void afficher(string = "");
    ~facture(void);
    void calculTotal();
    void afficherPrix();
};
```

100 %

Exemple facture

```
facture::facture(int cf,string libelle, int nbArticles)
{
    this->cf=cf;
    this->libelle=libelle;
    this->nbArticles=nbArticles;
    articles=new float[nbArticles];
    saisirPrix();
    calculTotal();
}

void facture::calculTotal()
{
    total=0;
    for(int i=0; i<nbArticles;i++)
        total+=*(articles+i);
}
```

```
void facture::saisirPrix()
{
    cout<<"\n saisir les prix "<<endl;
    for(int i=0; i<nbArticles; i++)
        cin>>*(articles+i);
}

facture::~~facture(void)
{
    delete[]articles;
}

void facture::afficherPrix()
{
    cout<<"\n affichage des prix "<<endl;
    for(int i=0; i<nbArticles; i++)
        cout<<*(articles+i)<<" ";
    cout<<endl;
}
```

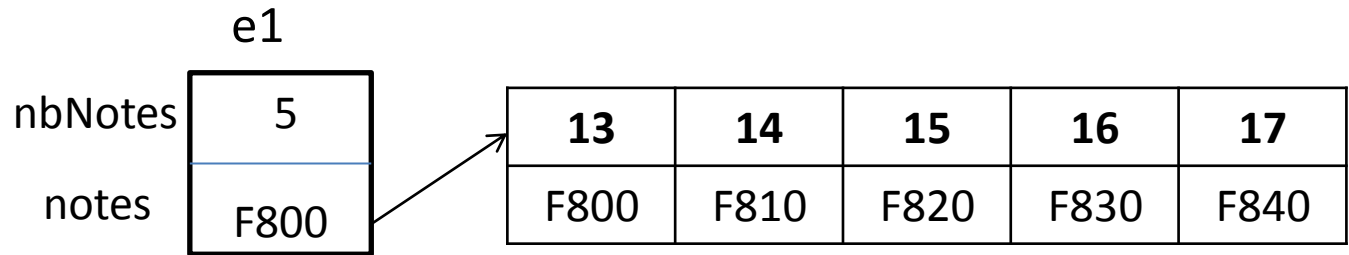
```
void facture::afficher(string msg)
{
    cout<<msg<<endl;
    cout<<"\n le code est "<<cf<<endl;
    cout<<"\n le libelle est "<<libelle<<endl;
    cout<<"\n le nbre d'articles "<<nbArticles<<endl;
    afficherPrix();
    cout<<"\n le total "<<total<<endl;
}
```


plan

1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Les membres données et fonctions statiques
6. Protection contre les inclusions multiples
7. Surdéfinition des fonctions membres
8. Arguments par défaut des fonctions membres
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. Objet retourné par une fonction
11. Autoréférence: le mot clé this
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

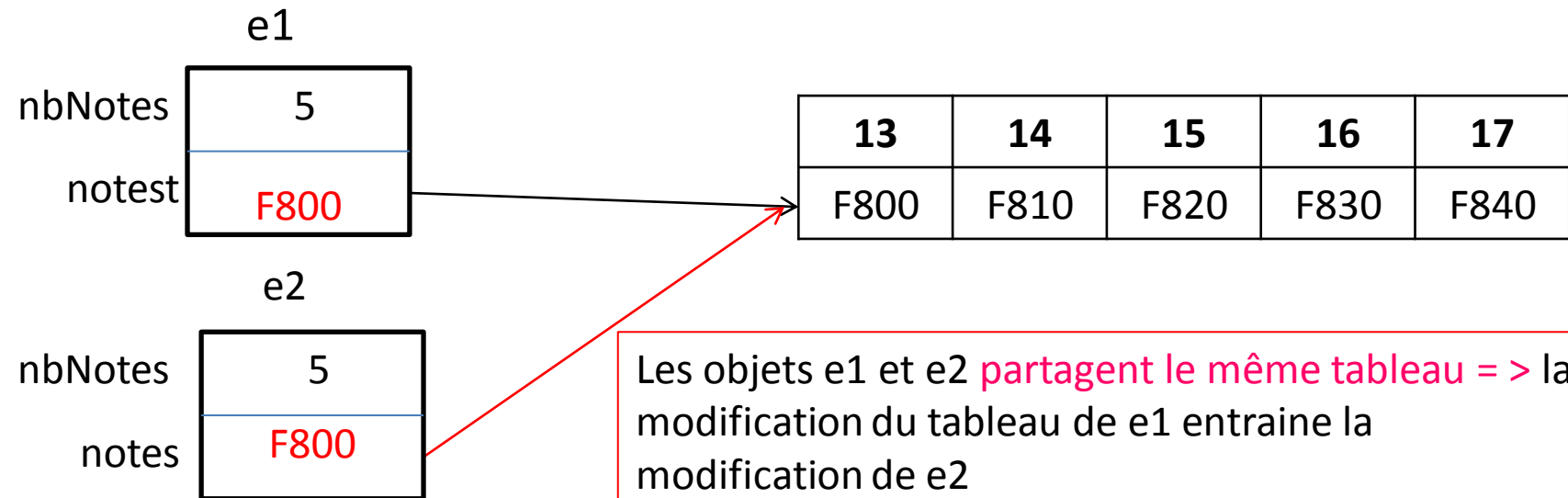
Problème: copie superficielle

```
class etudiant
{
    int nbNotes;
    int * notes;
public:
    // les méthodes
}
```



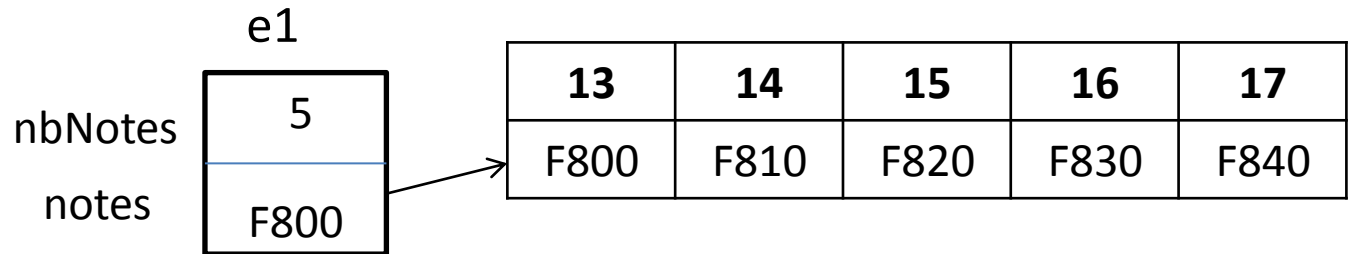
etudiant e1;
L'objet e1 contient une partie dynamique

e2=e1;

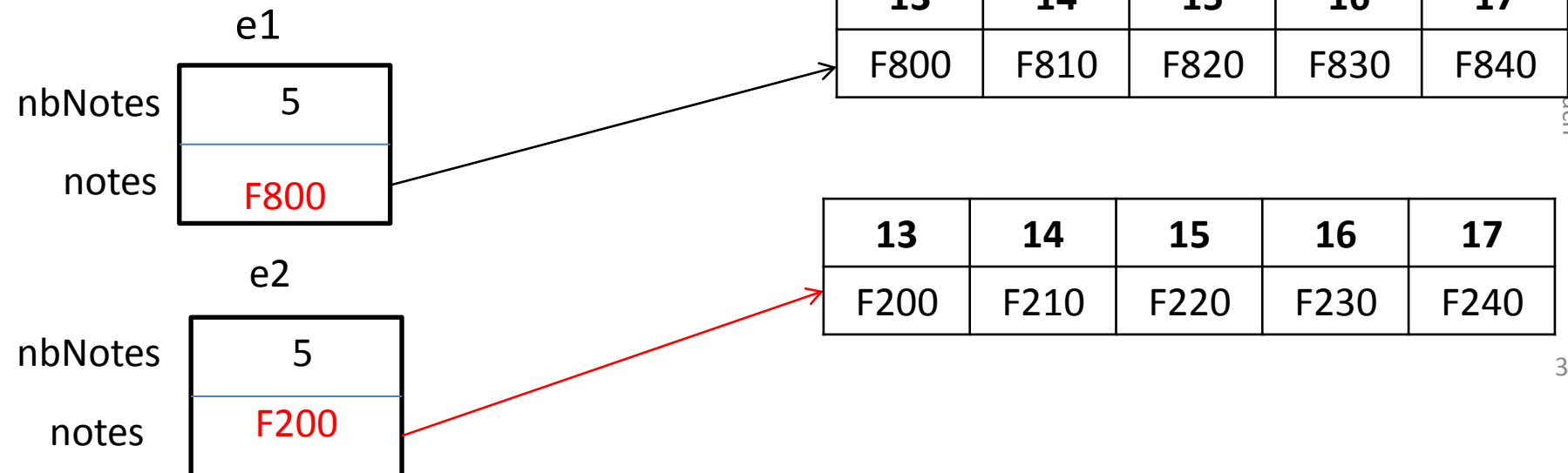


Solution: copie profonde

```
class etudiant
{
    int nbNotes;
    int * notes;
public:
    // les méthodes
}
```

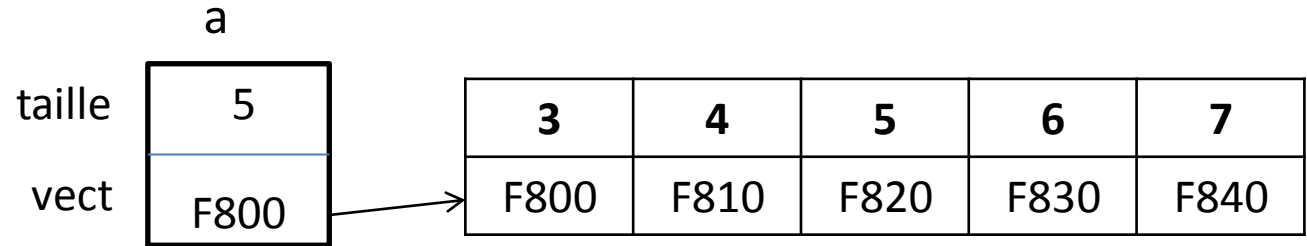


e2=e1



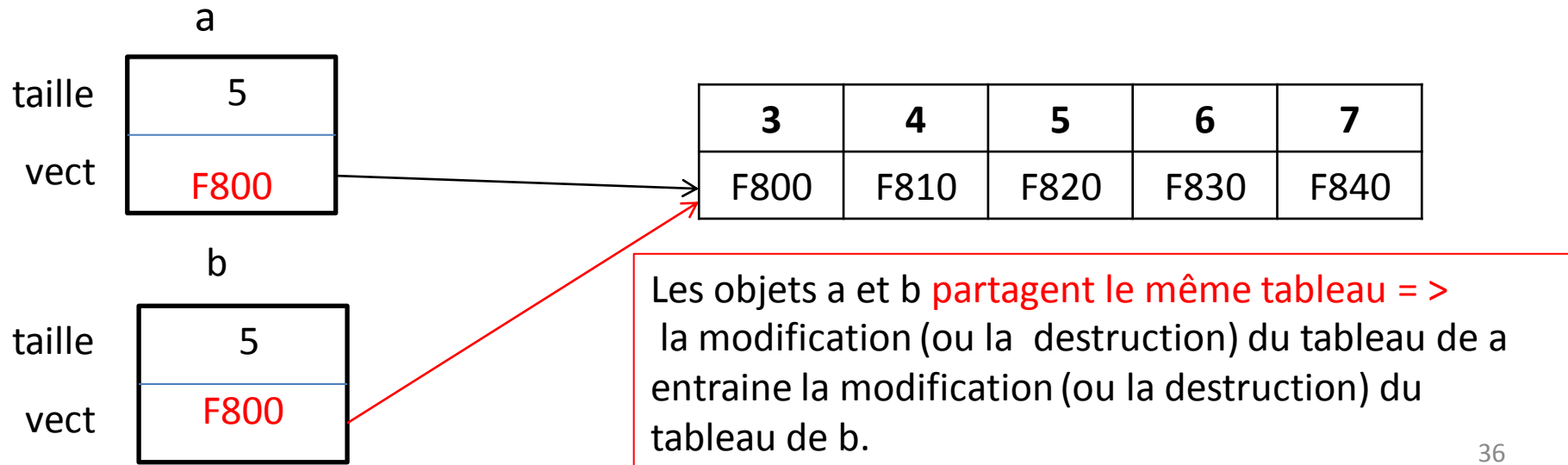
Problème: copie superficielle

```
class vecteur
{
    int taille;
    int * vect;
public:
    // les méthodes
}
```

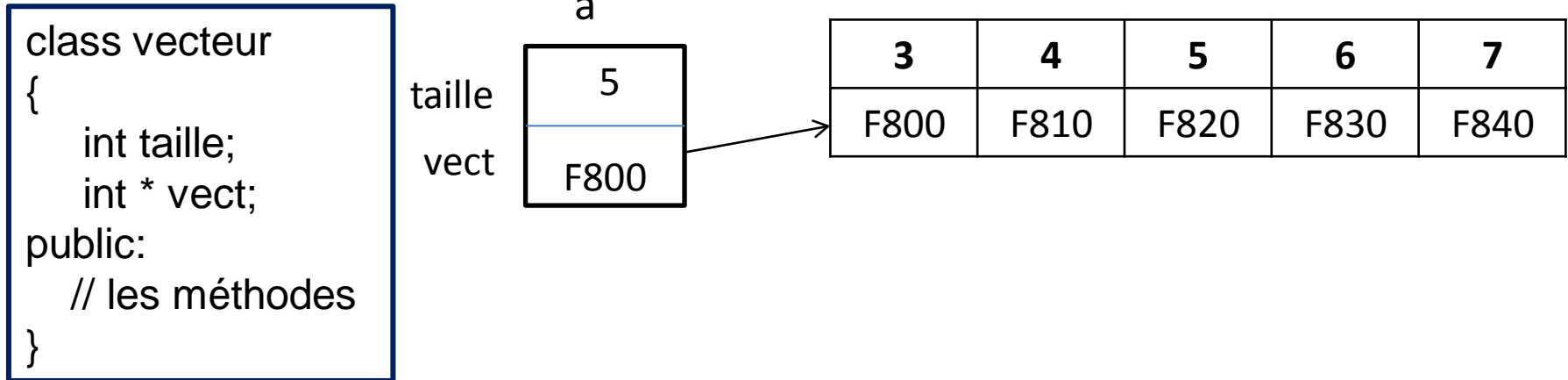


L'objet a contient un pointeur sur une partie dynamique

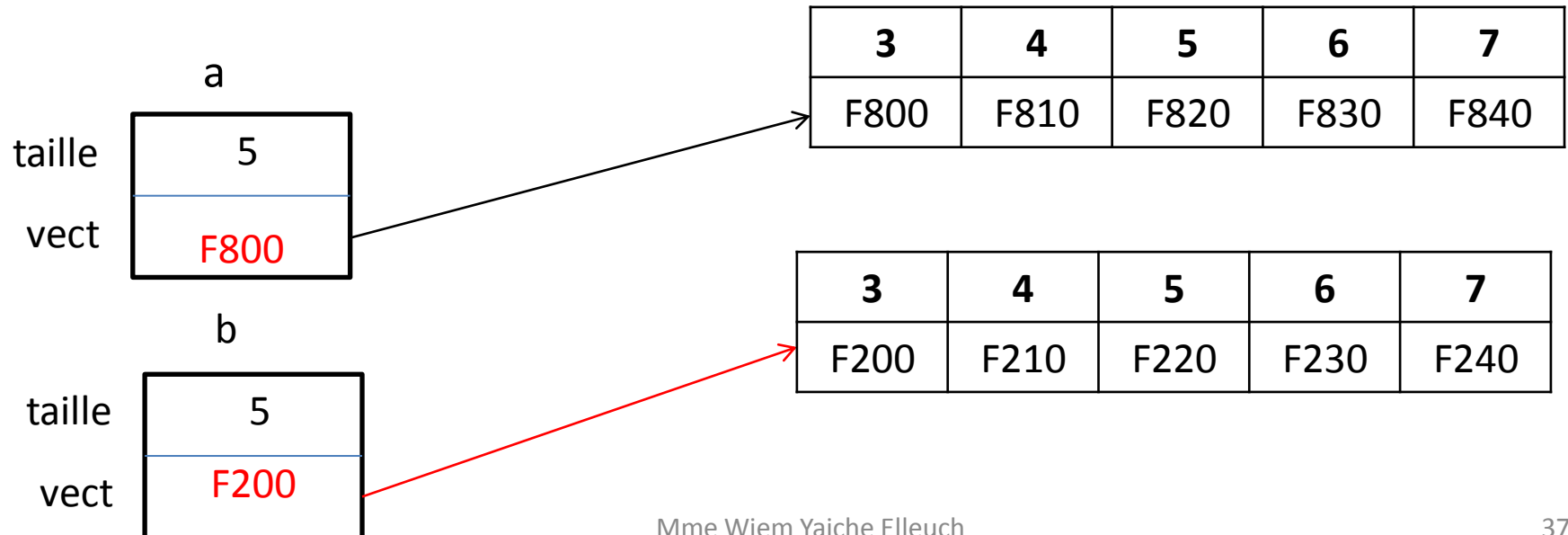
Création de b à partir de a ➔ objet b est une copie de l'objet a



Solution: copie profonde



Création de b à partir de a



Le constructeur de recopie

- Pour éviter le problème de recopie d'adresse, il faut munir la classe d'un constructeur par recopie (copy constructor). Ce dernier assure une **copie profonde**.
- Si aucun constructeur de recopie n'est défini, alors le compilateur crée un **constructeur de recopie par défaut**. Ce dernier assure une **copie superficielle**.
- Le constructeur de recopie dispose d'**un seul argument du type de la classe** et **transmis obligatoirement par référence** (classe T).

```
T::T (const T &)
```

Le constructeur de recopie

- Le **constructeur de recopie**, intervient dans les situations d'"**initialisation d'un objet**", c'est-à-dire lorsqu'il est nécessaire de réaliser une **copie d'un objet existant**.
- Une initialisation par recopie d'un objet est donc la **création d'un objet par recopie d'un objet existant de même type**.
- Il existe trois situations de ce type:
 - transmission de la valeur d'un objet en argument d'une fonction,
 - transmission de la valeur d'un objet en résultat d'une fonction,
 - initialisation d'un objet lors de sa déclaration par un objet de même type

Problèmes rencontrés lorsque l'objet contient un pointeur vers une partie dynamique

PROBLEME: copie superficielle

SOLUTION: copie profonde

Cas	problème	solution
Cas 1: Affectation de deux objets	Objets dépendants (partagent la même partie dynamique)	Surdéfinition de l'opérateur d'affectation = (→ chapitre surcharge des opérateurs)
Cas 2: Initialisation d'un objet lors de sa déclaration Création+initilisation	Objets dépendants (partagent la même partie dynamique)	Constructeur de recopie
Cas 3: Passage d'un objet par valeur en argument d'une fonction	Libération de la partie dynamique de l'objet à la sortie de la fonction	Constructeur de recopie
Cas 4: Objet retourné par une fonction	Libération de la partie dynamique de l'objet à la sortie de la fonction	Constructeur de recopie


```
#pragma once
#include<iostream>
using namespace std;
#include<string>
class etud
{
    int nbNotes;
    int*notes;
public:
    etud(const etud&);
    // constr de recopie
    etud(int=3);
    ~etud(void);
    void afficher(string="");
    bool similaire(etud);
    etud symetrique();
};
```

```
// bool res = a.similaire(b);
bool etud::similaire(etud e)
{
    cout<<"\n*****"<<endl;
    e.afficher("etud e");
    cout<<"\n*****"<<endl;
    if( e.nbNotes==nbNotes) return 1;
    return 0;
}
```

```
etud::etud(int n)
```

```
{
    nbNotes=n;
    notes=new int[n];
    for(int i=0; i<n; i++)
        cin>>notes[i];
}
```

```
void etud::afficher(string msg)
```

```
{
    cout<<msg<<endl;
    cout<<"\n nbre de notes "<<nbNotes<<endl;
    cout<<"\n la valeur de notes "<<notes<<endl;
    for(int i=0; i<nbNotes; i++)
        cout<<notes[i]<<"\t";
    cout<<endl;
}
```

```
etud::~~etud(void)
```

```
{
    delete []notes;
}
```

```
//etud b =a.symetrique();
```

```
etud etud::symetrique()
```

```
{
    etud e(nbNotes);
    cout<<"\n*****"<<endl;
    e.afficher("etud e");
    cout<<"\n*****"<<endl;
    return e;
}
```

```

// etud b=a;
// etud b(a);
etud::etud(const etud &w)
{
    cout<<"\n*****"<<endl;
    cout<<"\n appel  constr de recopie "<<this<<endl;

    nbNotes=w.nbNotes;
    notes=new int[nbNotes];
    for(int i=0; i<nbNotes ; i++)
        notes[i]=w.notes[i];
    cout<<"\n FIN constr recopie*****"<<endl;
}

```

```

// création d'un objet
// appel du constructeur
// appel du constr de recopie:
    // créer un objet copie (à partir) d'un objet existant
    // création + initialisation (etud b=a;)
    // passage par valeur d'un objet
    // objet retourné par une méthode

// il existe 2 catégories de classes
// classe sans partie dynamique==> constr de recopie par défaut
// classe avec partie dynamique: ==> construct de recopie

```

```

class vecteur
{
    int taille;
    int* vect;
public:
    vecteur(int =2);
    void remplir();
    void afficher(string = "");
    int produit_scalaire(vecteur);
    vecteur multiplier(int);
    ~vecteur(void);
};

```

```

vecteur::vecteur(int nb)
{
    taille=nb;
    vect=new int[taille];
}

```

```

void vecteur::remplir()
{
    cout<<"\n remplissage "<<endl;
    for(int i=0; i<taille; i++)
        cin>>vect[i];
}

```

```

void vecteur::afficher(string msg)
{
    cout<<msg<<endl;
    cout<<"\n la valeur de taille "<<taille<<endl;
    cout<<"\n la valeur de vect "<<vect<<endl;
    for(int i=0; i<taille; i++)
        cout<<vect[i]<<" ";
    cout<<endl;
}

```

```

vecteur::~~vecteur(void)
{
    delete[]vect;
}

```

```

int vecteur::produit_scalaire(vecteur v)
{
    int s=0;
    for(int i=0; i<taille; i++)
        s+=vect[i]*v.vect[i];
    return s;
}

```

```

vecteur vecteur::multiplier(int x)
{
    vecteur v(taille);
    for(int i=0; i<taille; i++)
        v.vect[i]=vect[i]*x;
    return v;
}

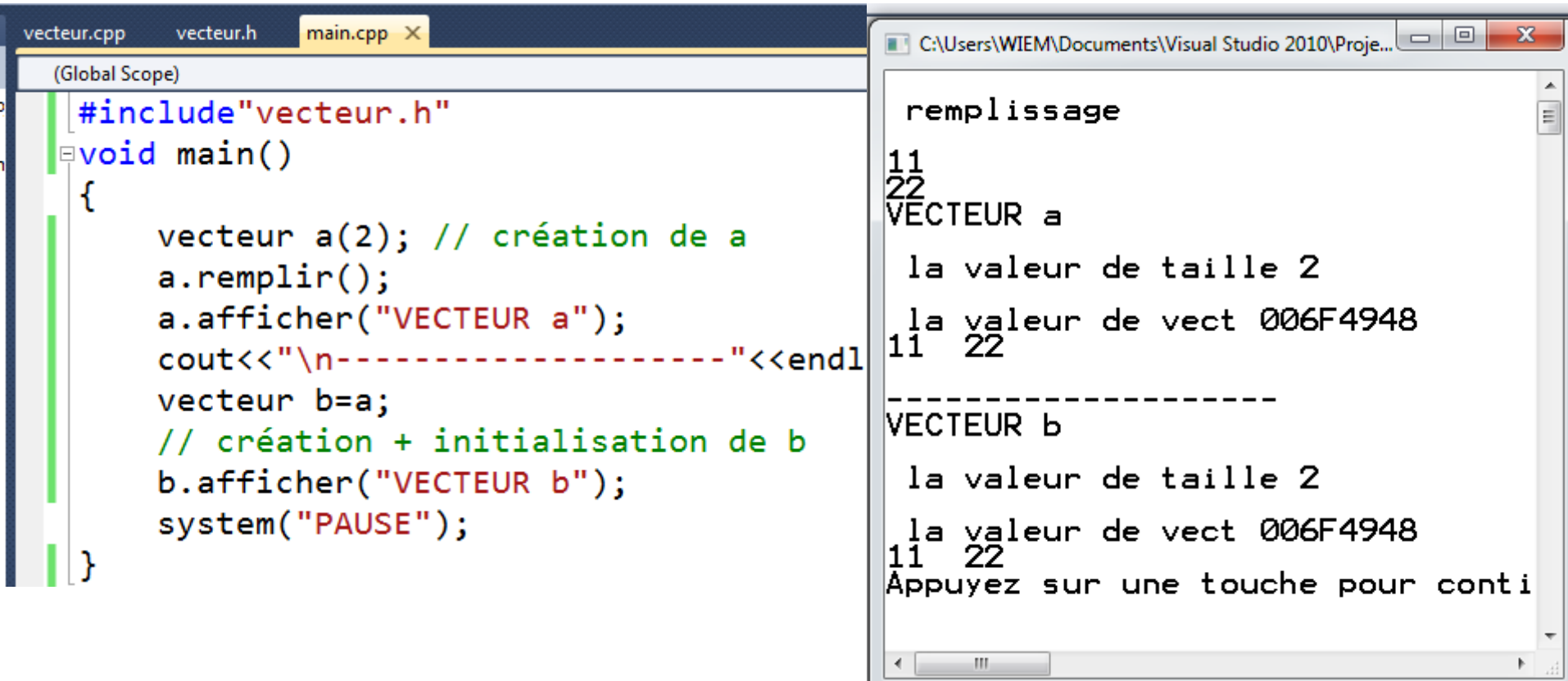
```

Cas 1: affectation

```
teur.cpp  vecteur.h  main.cpp* X
(Global Scope)
#include "vecteur.h"
void main()
{
    vecteur a(2); // création de a
    a.remplir();
    a.afficher("VECTEUR a");
    cout<<"\n-----"<<endl;
    vecteur b(1); // création de b
    b.remplir();
    b.afficher("VECTEUR b");
    cout<<"\n-----"<<endl;
    b=a; // a et b existent déjà
    b.afficher("VECTEUR b");
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingC2014\Deb...
remplissage
11
22
VECTEUR a
    la valeur de taille 2
    la valeur de vect 00554948
11  22
-----
remplissage
55
VECTEUR b
    la valeur de taille 1
    la valeur de vect 005549E0
55
-----
VECTEUR b
    la valeur de taille 2
    la valeur de vect 00554948
11  22
Appuyez sur une touche pour continuer...
```

Cas 2: Initialisation d'un objet lors de sa déclaration



The image shows a screenshot of the Visual Studio 2010 IDE. The left pane displays the source code for `main.cpp`, and the right pane shows the program's output.

Source Code (main.cpp):

```
#include "vecteur.h"
void main()
{
    vecteur a(2); // création de a
    a.remplir();
    a.afficher("VECTEUR a");
    cout<<"\n-----"<<endl;
    vecteur b=a;
    // création + initialisation de b
    b.afficher("VECTEUR b");
    system("PAUSE");
}
```

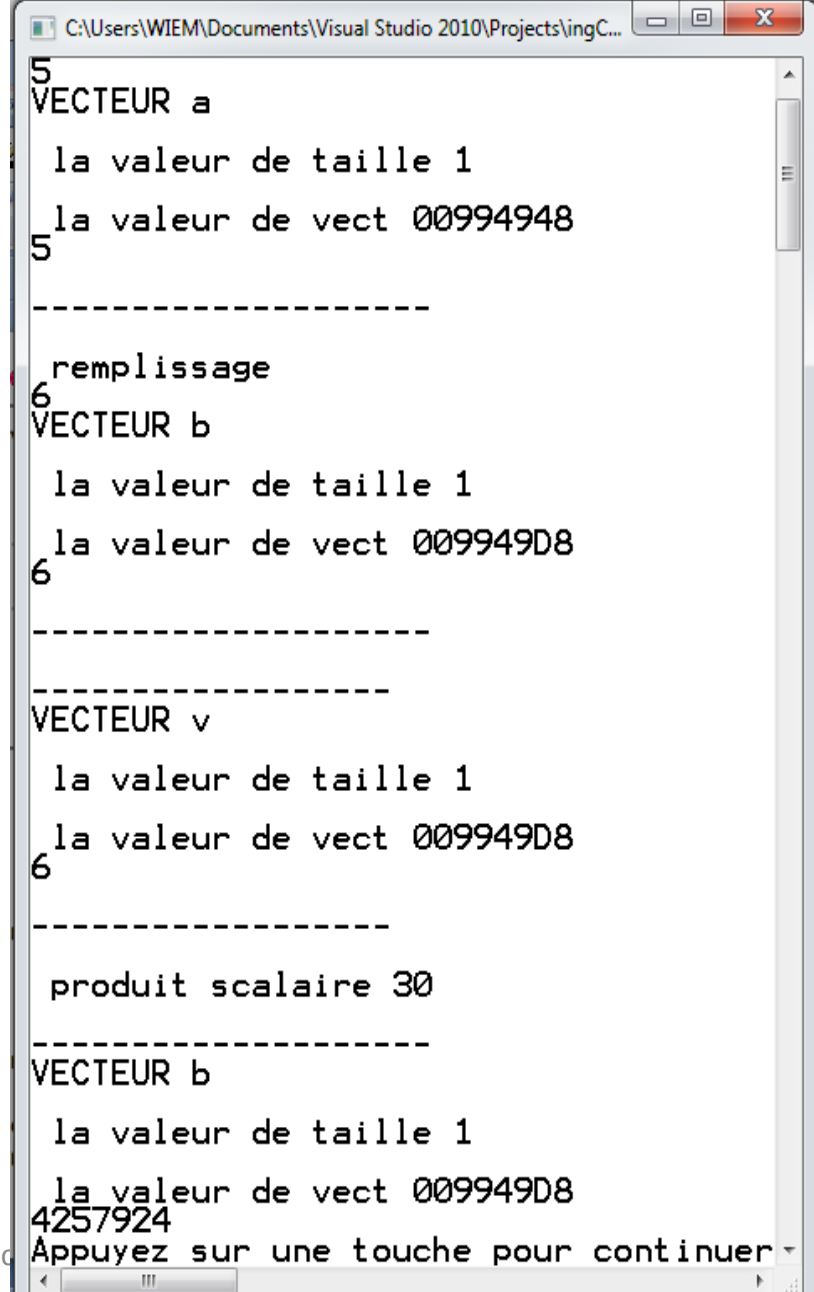
Output (Console):

```
remplissage
11
22
VECTEUR a
    la valeur de taille 2
    la valeur de vect 006F4948
11  22
-----
VECTEUR b
    la valeur de taille 2
    la valeur de vect 006F4948
11  22
Appuyez sur une touche pour conti
```

Cas 3: Passage d'un objet par valeur en argument d'une fonction

```
int vecteur::produit_scalaire(vecteur v)
{
    int s=0;
    cout<<"\n-----"<<endl;
    v.afficher("VECTEUR v");
    cout<<"\n-----"<<endl;
    for(int i=0; i<taille; i++)
        s+=vect[i]*v.vect[i];
    return s;
}
```

```
void main()
{
    vecteur a(1);
    a.remplir();
    a.afficher("VECTEUR a");
    cout<<"\n-----"<<endl;
    vecteur b(1);
    b.remplir();
    b.afficher("VECTEUR b");
    cout<<"\n-----"<<endl;
    int s=a.produit_scalaire(b);
    cout<<"\n produit scalaire "<<s<<endl;
    cout<<"\n-----"<<endl;
    b.afficher("VECTEUR b");
    system("PAUSE");
}
```



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\lingC...
5
VECTEUR a
    la valeur de taille 1
    la valeur de vect 00994948
5
-----
    remplissage
6
VECTEUR b
    la valeur de taille 1
    la valeur de vect 009949D8
6
-----
VECTEUR v
    la valeur de taille 1
    la valeur de vect 009949D8
6
-----
    produit scalaire 30
-----
VECTEUR b
    la valeur de taille 1
    la valeur de vect 009949D8
4257924
Appuyez sur une touche pour continuer
```

Cas 4: Objet retourné par une fonction

```
vecteur vecteur::multiplier(int x)
{
    vecteur v(taille);
    for(int i=0; i<taille; i++)
        v.vect[i]=vect[i]*x;
    cout<<"\n-----"<<endl;
    v.afficher("VECTEUR v");
    cout<<"\n-----"<<endl;
    return v;
}
```

vecteur.cpp* vecteur.h main.cpp X

(Global Scope)

```
#include "vecteur.h"
void main()
{
    vecteur a(2);
    a.remplir();
    a.afficher("VECTEUR a");
    cout<<"\n-----"<<endl;
    vecteur b=a.multiplier(3);
    b.afficher("VECTEUR b");
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingC20...

```
remplissage
11
22
VECTEUR a
    la valeur de taille 2
    la valeur de vect 00754948
11 22
-----
VECTEUR v
    la valeur de taille 2
    la valeur de vect 007549E0
33 66
-----
VECTEUR b
    la valeur de taille 2
    la valeur de vect 007549E0
2226968 0
Appuyez sur une touche pour continuer.
```

```
vecteur.h* x etudiant.cpp etudiant.h main.cpp
(Global Scope)
#pragma once
#include<iostream>
using namespace std;
#include<string>
class vecteur
{
    int taille;
    int* vect;
public:
    vecteur(int =2);
    vecteur (const vecteur &);
    void remplir();
    void afficher(string = "");
    int produit_scalaire(vecteur);
    vecteur multiplier(int);
    ~vecteur(void);
};
```

```
vecteur.cpp* x vecteur.h* etudiant.cpp etudiant.h main.cpp
vecteur
vecteur::vecteur(const vecteur &w)
{
    cout<<"\n +++ constr de recopie "<<endl;
    cout<<this<<endl;
    taille=w.taille;
    vect=new int[taille];
    for(int i=0; i<taille; i++)
        vect[i]=w.vect[i];
    cout<<"\n +++ FIN constr de recopie "<<endl;
}
```



```

vecteur.cpp  vecteur.h  main.cpp x
(Global Scope)
void main()
{
    vecteur a(2);
    a.remplir();
    a.afficher("VECTEUR a");
    cout<<"\n-----"<<endl;
    vecteur b=a;
    cout<<"\n-----"<<endl;
    b.afficher("VECTEUR b");
    system("PAUSE");
}

```

- L'objet b (de main) appelle le constructeur de copie
- w (l'argument du constr de copie) est une référence sur l'objet a (de main)
- **const** dans l'argument du constructeur permet de protéger l'objet a contre d'éventuelles modifications.

```

vecteur.cpp x  vecteur.h  main.cpp
vecteur
vecteur::vecteur(const vecteur &w)
{
    cout<<"\n +++ constr de copie " <<endl;
    cout<<this<<endl;
    cout<<"\n l'adresse de w " <<&w<<endl;
    taille=w.taille;
    vect=new int[taille];
    for(int i=0; i<taille; i++)
        vect[i]=w.vect[i];
    cout<<"\n +++ FIN constr de copie " <<endl;
}

```

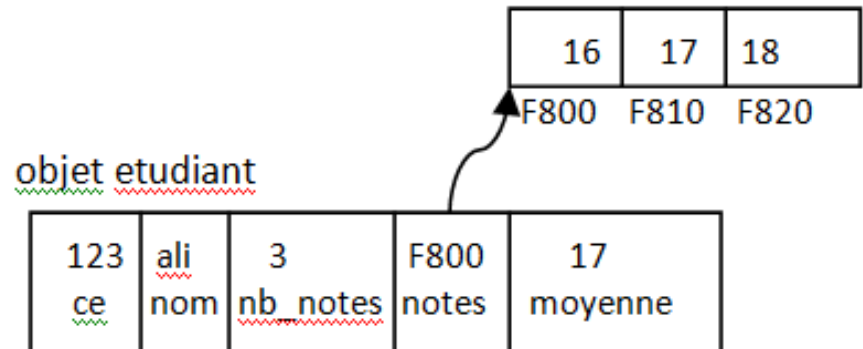
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\j...
remplissage
22
33
VECTEUR a  003CFC90
la valeur de taille 2
la valeur de vect 00254948
22 33
-----
+++ constr de copie
003CFC90
l'adresse de w 003CFC90
+++ FIN constr de copie
-----
VECTEUR b  003CFC90
la valeur de taille 2
la valeur de vect 002549E0
22 33
Appuyez sur une touche pour continu

```

Exemple 2

```
etudiant.cpp  etudiant.h  main.cpp
etudiant
class etudiant
{
    int ce;
    string nom;
    int nb_notes;
    float*notes;
    float moyenne;
public:
    etudiant( const etudiant &);
    etudiant(int =999, string ="", int =2);
    void saisir_notes();
    void afficher_notes();
    void calcul_moyenne();
    void afficher(string = "");
    // setteurs getteurs
    ~etudiant(void);
};
```



Exemple 3

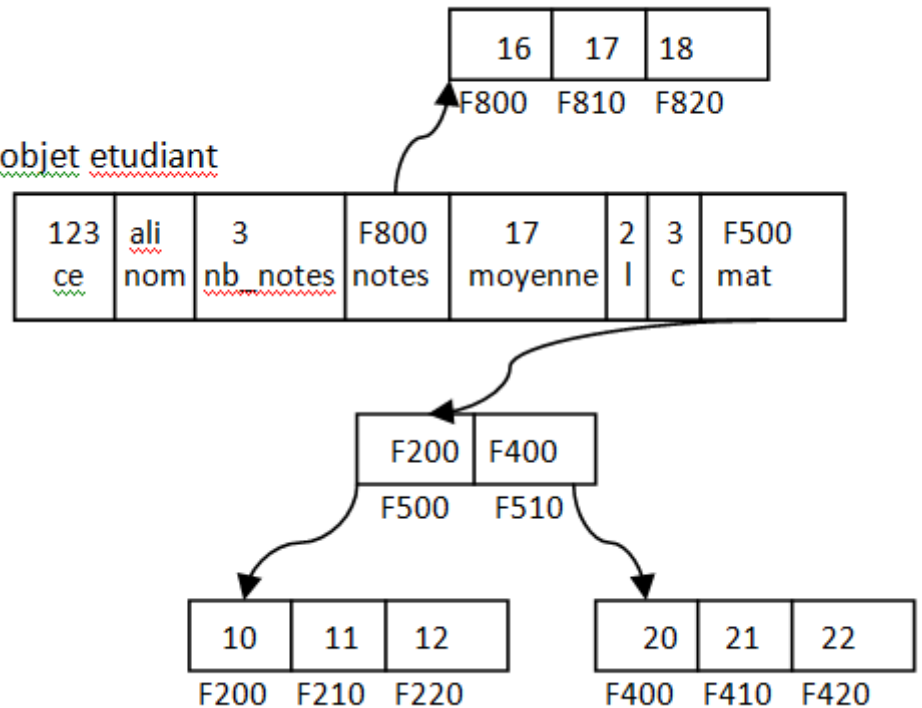
```

etudiant.cpp*  etudiant.h*  main.cpp
etudiant

class etudiant
{
    int ce;
    string nom;
    int nb_notes;
    float*notes;
    float moyenne;
    int l;
    int c;
    int**mat;
public:
    etudiant( const etudiant &);
    etudiant(int =999, string ="", int =2);
    void saisir_notes();
    void afficher_notes();
    void calcul_moyenne();
    void afficher(string = "");
    // setteurs getteurs
    ~etudiant(void);
};

```

objet etudiant



Résumé: Création d'un objet

- Par déclaration
 - Appel du constructeur
 - Appel du constructeur de copie
- Par allocation (avec **new**)
 - Appel du constructeur
 - Appel du constructeur de copie

Exemple: Création d'un point

- Par déclaration
 - Appel du constructeur
 - Appel du constructeur de copie
- Par allocation
 - Appel du constructeur
 - Appel du constructeur de copie

Création d'un point par déclaration avec appel du constructeur

```
void main()
{
    point a;
    a.afficher();
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingA2014\Debu...
003BF7BC coordonnees: 99 88
Appuyez sur une touche pour continuer...

```
void main()
{
    point a(5,6);
    a.afficher();
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingA2014\D...
0035F960 coordonnees: 5 6
Appuyez sur une touche pour continuer...

Exemple: Création d'un point

- Par déclaration
 - Appel du constructeur
 - Appel du constructeur de copie
- Par allocation
 - Appel du constructeur
 - Appel du constructeur de copie

Création d'un point par déclaration avec appel du constructeur de copie:

Initialisation d'un objet à partir d'un autre objet

```
void main()
{
    point a(6,8);
    point b=a;
    b.afficher();
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\De...
003DF70C coordonnees 6 8
Appuyez sur une touche pour continuer.

```
void main()
{
    point a(6,8);
    point b(a);
    b.afficher();
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\te...
0034FD90 coordonnees 6 8
Appuyez sur une touche pour continuer...

```
void main()
{
    point b=point(6,8);
    b.afficher();
    system("PAUSE");
}
```

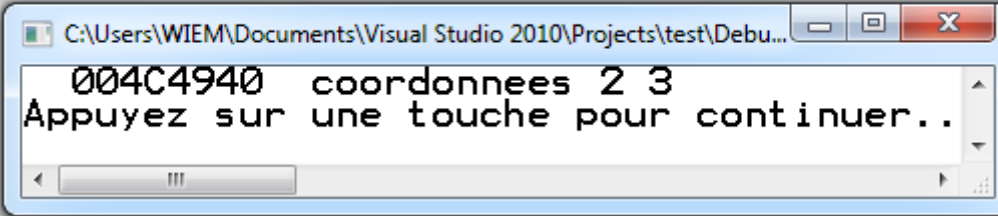
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\De...
003CFA70 coordonnees 6 8
Appuyez sur une touche pour continuer.

Exemple: Création d'un point

- Par déclaration
 - Appel du constructeur
 - Appel du constructeur de copie
- Par allocation
 - Appel du constructeur
 - Appel du constructeur de copie

Création d'un point par allocation avec appel du constructeur

```
void main()
{
    point *q=new point(2,3);
    // appel du constructeur
    q->afficher();
    system("PAUSE");
}
```

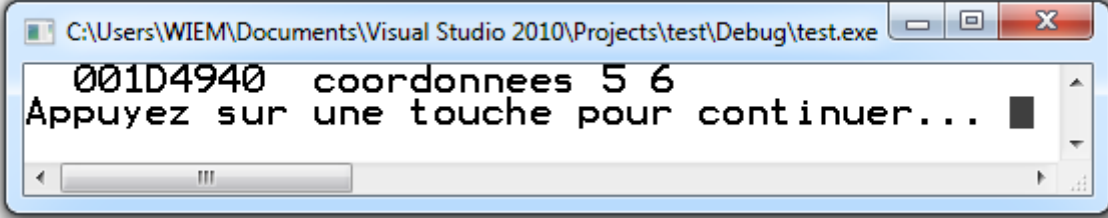


Exemple: Création d'un point

- Par déclaration
 - Appel du constructeur
 - Appel du constructeur de copie
- Par allocation
 - Appel du constructeur
 - Appel du constructeur de copie

Création d'un point par allocation avec appel du constructeur de copie:

```
void main()
{
    point a(5,6);
    point *q=new point(a);
    // l'argument est un objet: a
    // appel du constructeur de copie par défaut
    // q contient l'adresse du nouvel objet créé
    // le nouvel objet créé est une copie de a
    q->afficher();
    system("PAUSE");
}
```



plan

1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
4. Notion de constructeur et de destructeur
5. Les membres données et fonctions statiques
6. Protection contre les inclusions multiples
7. Surdéfinition des fonctions membres
8. Arguments par défaut des fonctions membres
9. Cas des objets transmis en argument d'une fonction (par valeur, par adresse, par référence)
10. Objet retourné par une fonction
11. Autoréférence: le mot clé this
12. Constructeur de copie
13. Objets membres
14. Tableau d'objets

Objets membres: mise en œuvre des constructeurs et des destructeurs

- D'une manière générale, la situation d'objets membres correspond à une relation entre classes du type **relation de possession** ("relation **a**" - du verbe avoir).
- exemples:
 - Un cercle **possède (a)** un centre
 - Une voiture **possède (a)** un moteur
 - Un étudiant **possède (a)** une date de naissance

exemple

```
class cercle
{
    point centre;
    float rayon;
public:
    cercle(int =2,int =3,float =4.4);
    void afficher(string = "");
    ~cercle(void);
};
```

spécifier les arguments à fournir au constructeur de *point* → ceux-ci doivent être choisis obligatoirement parmi les arguments fournis à *cercle*

cercle.cpp* X cercle.h* main.cpp

(Global Scope)

```
#include "cercle.h"
cercle::cercle(int abs, int ord, float r):centre(abs,ord)
{
    cout<<"\n appel constr cercle " <<this<<endl;
    rayon=r;
}
void cercle::afficher(string msg)
{
    cout<<msg<<" " <<this<<endl;
    centre.afficher();
    cout<<"\n le rayon est " <<rayon<<endl;
}
cercle::~~cercle(void)
{
    cout<<"\n appel destr cercle " <<endl;
}
```

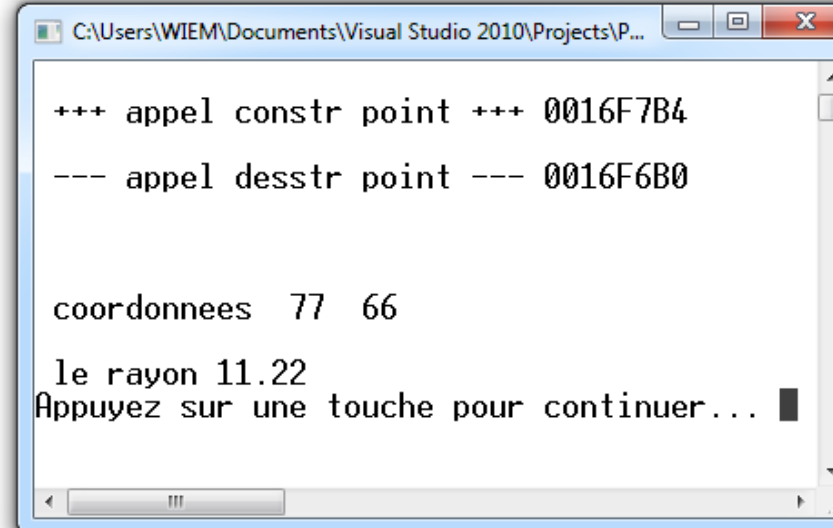
Constructeur de cercle avec un argument de type point

```
#include "point.h"
class cercle
{
    point centre; // objet centre de type point// objet membre
    float rayon;
public:
    cercle(int =11,int =22, float=33.33);
    cercle(point,float);
    ~cercle(void);
    void afficher(string = "");
};
```

```
cercle::cercle(point pt, float r):centre(pt), rayon(r)
{
    |
}
```

```
void main()
{
    point a(77,66);
    cercle c(a,11.22);
    c.afficher();

    system("PAUSE");
}
```



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\P...
+++ appel constr point +++ 0016F7B4
--- appel destr point --- 0016F6B0

coordonnees  77  66

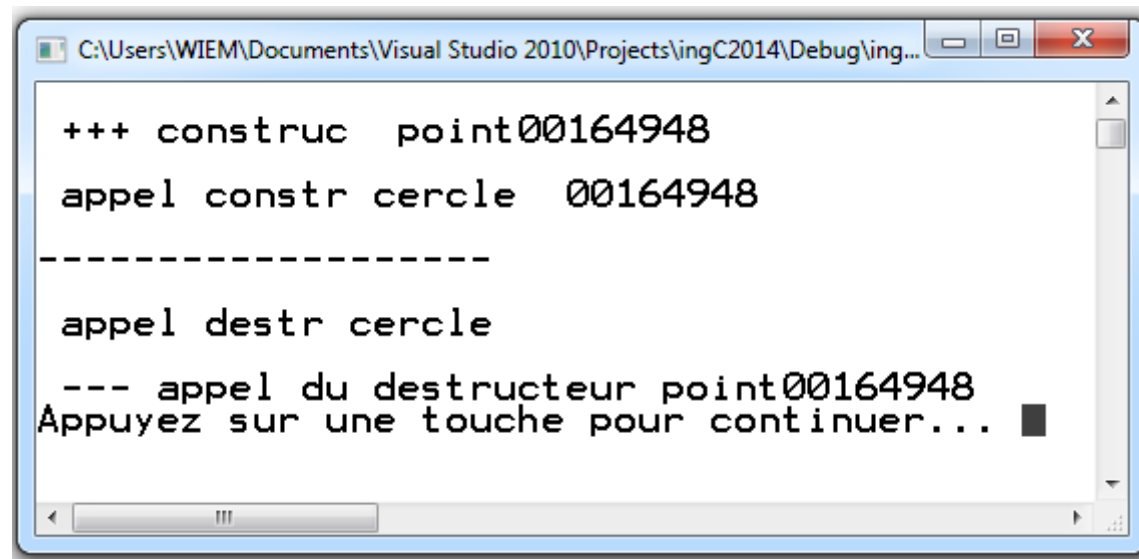
le rayon 11.22
Appuyez sur une touche pour continuer...
```


Objets membres: mise en œuvre des constructeurs et des destructeurs

- Si la classe point possède un constructeur ou plus, alors la classe cercle doit obligatoirement en posséder au moins un.
- Les constructeurs seront appelés dans l'ordre suivant: point, cercle.
- S'il existe des destructeurs, ils seront appelés dans l'ordre inverse.

Ordre de l'appel des constructeurs et des destructeurs

```
void main()
{
    cercle *q= new cercle(1,2,3);
    cout<<"\n-----"<<endl;
    delete q;
    system("PAUSE");
}
```



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingC2014\Debug\ing...

+++ construc point00164948
appel constr cercle 00164948
-----

appel destr cercle

--- appel du destructeur point00164948
Appuyez sur une touche pour continuer...
```

Remarque 1

- Si la classe *point* dispose d'un constructeur **sans arguments**, alors le constructeur de cercle peut s'écrire:

```
cercle::cercle(int abs,int ord, float r)
{
    rayon=r;
    centre = point(abs,ord);
}
```

Remarque 2

- Dans le cas d'objets comportant **plusieurs objets membres**, la sélection des arguments destinés aux différents constructeurs se fait en séparant chaque liste par **une virgule**.

```
class A
{
    .....
    // constructeur
    A(int);
};
```

```
class B
{
    .....
    // constructeur
    B(float, int);
};
```

```
class C
{
    A a;
    B b;
    .....
    // constructeur
    C(int,int,float);
    C(A,B); };
```

```
// constructeur de la classe C
C::C (int x, int y, float z): a(x), b(z,y)
{ ..... }
```

- le constructeur C ne sera exécuté qu'après les deux autres (l'ordre des imbrications est toujours respecté)

Exemple: la classe cercle possède 2 attributs objets: centre et dateCreation

```
date.h  etudian.cpp  etudiant.cpp  etudiant.h  main.cpp
(Global Scope)
class date
{
    int jour;
    int mois;
    int annee;
public:
    date(int =2,int =2 ,int =2017);
    ~date(void);
    void afficher(string = "");
};
```

```
date.cpp*  date.h  etudian.cpp  etudiant.cpp  etudiant.h  main.cpp
date
~date(void)
date::date(int j, int m, int a):jour(j), mois(m), annee(a)
{
    cout<<"\n appel constr date "<<this<<endl;
}
date::~~date(void)
{
    cout<<"\n appel destr date "<<this<<endl;
}
void date::afficher(string msg)
{
    cout<<msg<<endl;
    cout<<jour<<"/"<<mois<<"/"<<annee<<endl;
}
```

```

class cercle
{
    point centre; // objet membre
    date dateCreation; // objet membre
    float rayon;
public:
    cercle(int =55,int =55, float =55.55, int =2, int=2,int =2017);
    cercle(point ,date, float=55.55);
    ~cercle(void);
    void afficher(string = "");
};

```

```

cercle::cercle(int abs, int ord, float r,int j, int m, int a):
    centre(abs,ord), rayon(r), dateCreation(j,m,a)
{
    cout<<"\n appel constr cercle "<<this<<endl;
}
cercle::cercle( point p,date d, float r): centre(p), dateCreation(d), rayon(r)
{
    cout<<"\n appel constr cercle "<<this<<endl;
}

```

```
cercle.cpp* x cercle.h date.cpp date.h etudian.cpp etudiant.cpp etudiant.h main.cpp
(Global Scope)
cercle::~~cercle(void)
{
    cout<<"\n appel destr cercle "<<this<<endl;
}

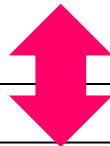
void cercle::afficher(string msg)
{
    cout<<msg<<endl;
    centre.afficher();
    dateCreation.afficher();
    cout<<"\n le rayon "<<rayon<<endl;
}
```

```
void main()
{
    point p;
    date d;
    cercle c(p,d,11.11);
    cout<<"\n-----"<<endl;
    c.afficher();
}
```

Initialisation de membres dans l'en-tête d'un constructeur

- Les attributs d'un objet peuvent être initialisés dans l'entête du constructeur, même s'ils sont d'un type de base.

```
point::point(int abs,int ord)
{
    cout<<"\n +++constructeur point"<<this<<endl;
    x=abs;
    y=ord;
}
```



```
point::point(int abs,int ord):x(abs),y(ord)
{
    cout<<"\n +++constructeur point"<<this<<endl;
}
```

```
point::point(int x, int y)
{
    cout<<"\n appel constr point "<<this<<endl;
    this->x=x;
    this->y=y;
}
```


Example 2

```
class etudiant
{
    int code;
    string nom;
    int nb_notes;
    float *notes;
    float moyenne;
public:
    etudiant(int =999,string ="",int =2);
    void saisir_notes();
    void afficher_notes();
    void calcul_moyenne();
    void afficher(string = "");
    etudiant(const etudiant&);
    ~etudiant(void);
};
```

```
etudiant::etudiant(int c, string n, int nb)
{
    int i,j;
    code=c;
    nom=n;
    nb_notes=nb;
    notes= new float[nb_notes];
    saisir_notes();
    calcul_moyenne();
}
```



```
etudiant::etudiant(int c, string n, int nb):code(c),nom(n),nb_notes(nb)
{
    int i,j;
    notes= new float[nb_notes];
    saisir_notes();
    calcul_moyenne();
}
```

Exemple 3

```
cercle::cercle(int abs, int ord, float r):centre(abs,ord), rayon(r)
{
    // initialisation de l'attribut centre dans l'entête du constructeur
    // initialisation de l'attribut rayon dans l'entête du constructeur
    // l'attribut centre est un objet
    // l'attribut rayon est d'un type de base (float)
    cout<<"\n+++ constructeur cercle+++   "<<this<<endl;
}
```