

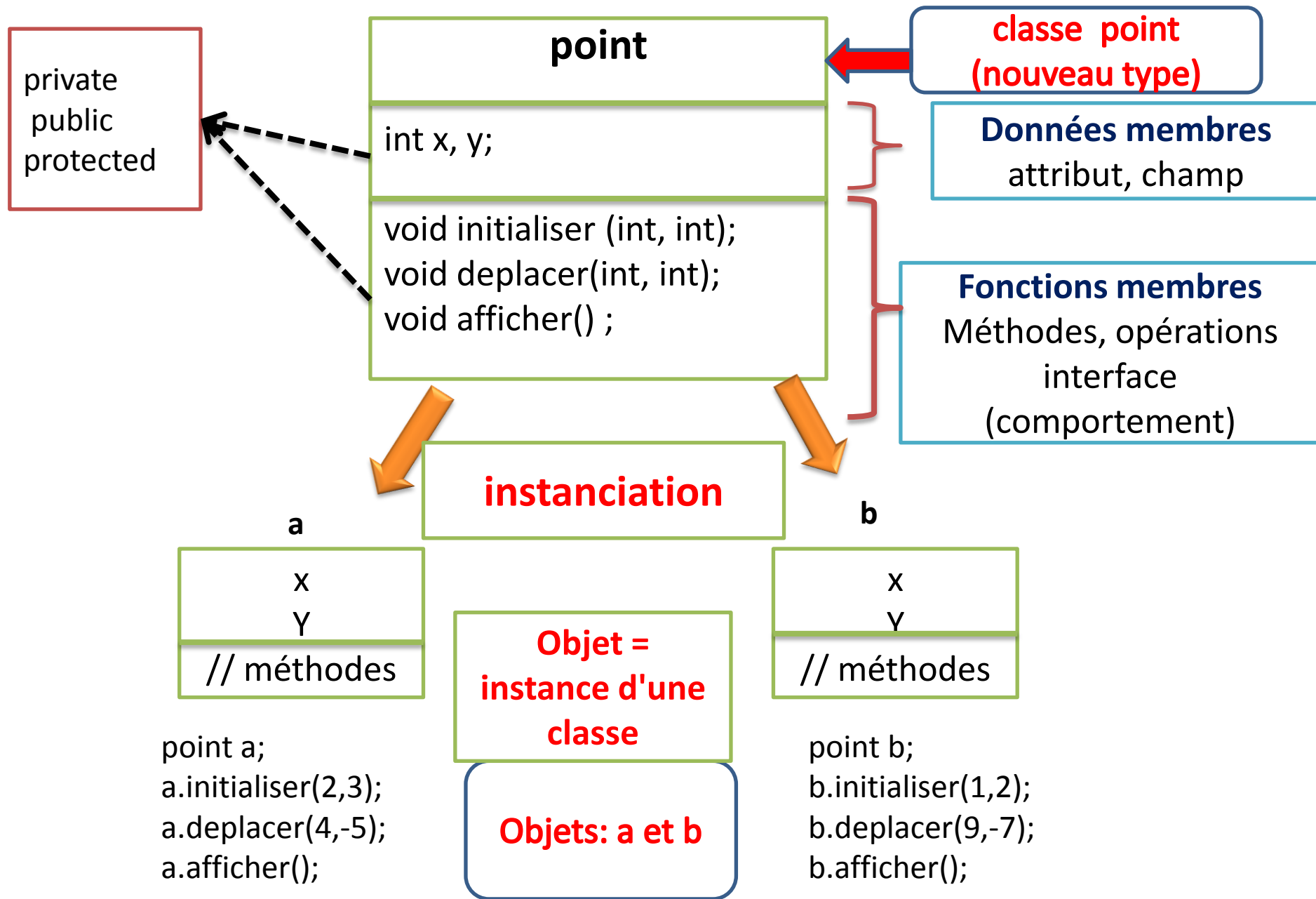
PОО – Langage C++

Les classes et les objets (partie 1)

1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

2018 - 2019



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

les possibilités de P.O.O. de C++,

- La P.O.O. reposent sur le concept de **classe**.
- *Une* classe est la généralisation de la notion de **type** défini par l'utilisateur
- Dans une classe se trouvent associées à la fois des données (**membres données**) et des méthodes (**fonctions membres**).
- En P.O.O. "pure", les données sont encapsulées et leur accès ne peut se faire que par le biais des méthodes
- Le type structure du C++ (mot clé **struct**) correspond à un cas particulier de la classe; il s'agit en effet d'une classe dans laquelle **aucune donnée n'est encapsulée**.

Déclaration d'une structure comportant des méthodes

```
struct point
{
    /* déclaration "classique" des données */
    int x;
    int y;
    /* déclaration des fonctions membre (méthodes); */
    void initialise (int, int) ;
    void deplace (int, int) ;
    void affiche () ;
};
```

```
void point::initialiser (int abs, int ord)
{
    x = abs ; y = ord ;
}
```

- Le symbole **::** correspond à ce que l'on nomme l'opérateur de "**résolution de portée**", lequel sert à modifier la portée d'un identificateur.
- Ici, il signifie que l'identificateur *initialise* concerné est celui défini dans *point*.

```
point.h x point.cpp main.cpp
point
#pragma once
struct point
{
    int x;
    int y;
    void initialise (int, int) ;
    void deplace (int, int) ;
    void affiche () ;
};
```

point est déclaré comme
une structure

```
point.h point.cpp x main.cpp
(Global Scope)
#include <iostream>
#include "point.h"
using namespace std ;
void point::initialise (int abs, int ord)
{
    x=abs ; y=ord ;
}

void point::deplace (int dx, int dy)
{ x+=dx;    y+=dy;
}

void point::affiche ()
{ cout<<"\n coordonnes: "<<x<<" "<<y<<"\n" ;
}
```

La fonction initialise opère
sur la variable a

Le mot struct
est facultatif
en C++

point.h point.cpp

(Global Scope)

```
#include <iostream>
#include "point.h"
using namespace std
void main()
```

```
point a;
a.initialise(2,3);
a.affiche();
a.deplace(4,-5);
a.affiche();
cout<<"\n affichage des valeurs de x et y: ";
cout<<a.x <<" " <<a.y<<endl<<endl;
system("PAUSE");
}
```

main()

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\...
coordonnes: 2 3
coordonnes: 6 -2
affichage des valeurs de x et y: 6 -2
Appuyez sur une touche pour continuer...
```

On peut accéder aux membres x et y de point à partir de main
(a.x, a.y)

Le principe d'encapsulation des données n'est pas respecté

a

x

y

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

Notion de classe

- en *C++*, la structure est un cas particulier de la classe.
- La déclaration d'une classe est voisine de celle d'une structure. En effet, il suffit
 - de remplacer le mot clé *struct* par le mot clé *class*,
 - de préciser quels sont les membres publics (fonctions ou données) et les membres privés en utilisant les mots clés *public* et *private*.

The image shows a side-by-side comparison of C++ code declarations in a code editor. The left window shows a `class point` declaration with `private` members and `public` methods. A callout box points to the `private` section with the text "Membres privés → encapsulation des données". The right window shows a `struct point` declaration with `public` members and `public` methods. A callout box points to the `public` section with the text "Par défaut private".

```
#pragma once
class point
{
private:
    int x;
    int y;
public:
    void initialise (int, int) ;
    void deplace (int, int) ;
    void affiche () ;
};
```

Membres privés →
encapsulation des
données

```
#pragma once
struct point
{
    int x;
    int y;
public:
    void initialise (int, int) ;
    void deplace (int, int) ;
    void affiche () ;
};
```

Par défaut
private

```
void main()
```

```
{
```

```
    point a;
```

```
    a.initialise(2,3);
```

```
    a.affiche();
```

```
    a.deplace(4,-5);
```

```
    a.affiche();
```

```
    cout<<"\n affichage des valeurs de x et y: ";
```

```
    cout<<a.x <<" " <<a.y<<endl<<endl;
```

```
    system("PAUSE");
```

```
}
```

a.initialise(2,3) est correcte
puisque initialise est
déclarée **public**

a.x et a.y ne sont plus
correctes puisque x et y
sont déclarées **private**

100 %

Output

Show output from: Build

```
io 2010\projects\seance1\seance1\main.cpp(12): error C2248: 'point::x' : cannot access private member declared in class 'point'  
/visual studio 2010\projects\seance1\seance1\point.h(4) : see declaration of 'point::x'  
/visual studio 2010\projects\seance1\seance1\point.h(3) : see declaration of 'point'  
io 2010\projects\seance1\seance1\main.cpp(12): error C2248: 'point::y' : cannot access private member declared in class 'point'  
/visual studio 2010\projects\seance1\seance1\point.h(5) : see declaration of 'point::y'  
/visual studio 2010\projects\seance1\seance1\point.h(3) : see declaration of 'point'
```

```
point.h  point.cpp*  main.cpp
(Global Scope)
#include <iostream>
#include "point.h"
using namespace std ;
void point::initialise (int abs, int ord)
{
    x=abs ; y=ord ;
}

void point::deplace (int dx, int dy)
{ x+=dx;    y+=dy;
}

void point::affiche ()
{cout<<"\n coordonnes: "<<x<<" "<<y<<"\n" ;
}
```

```
point.h  point.cpp*  main.cpp
(Global Scope)
void main()
{
    point a;
    a.initialise(2,3);
    a.affiche();
    a.deplace(4,-5);
    a.affiche();
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\sean...
coordonnes: 2 3
coordonnes: 6 -2
Appuyez sur une touche pour continuer
```

a est un instance de la classe point
a est un objet de type point

```
point.h x point.cpp main.cpp nt.h point.cpp
point (Global Scope)
#pragma once
class point
{
public:
    int x;
    int y;
public:
    void initialise (int,
    void deplace (int, in
    void affiche () ;
};

void main()
{
    point a;
    a.initialise(2,3);
    a.affiche();
    a.deplace(4,-5);
    a.affiche();
    cout<<"\n affichage des valeurs de x et y: ";
    cout<<a.x <<" " <<a.y<<endl<<endl;
    system("PAUSE");
}
```

a.x et a.y sont correctes
puisque x et y sont
déclarés public

Pas d'encapsulation des
données

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\seance1\Debug\...
coordonnees : 2 3
coordonnees : 6 -2
affichage des valeurs de x et y : 6 -2
Appuyez sur une touche pour continuer...
```

Remarque

- En général, on cherchera à respecter le principe **d'encapsulation des données**, même si on prévoit des fonctions membres appropriées pour y accéder.
- Dans cet exemple, toutes les fonctions membres étaient publiques.
- Il est tout à fait possible d'en rendre certaines privées. Dans ce cas, de telles fonctions ne seront plus accessibles de l'"extérieur" de la classe. Elles ne pourront être appelées que par d'autres fonctions membres.
- Les mots clés *public* et *private* peuvent apparaître à plusieurs reprises dans la définition d'une classe
- Si aucun de ces deux mots n'apparaît dans la définition d'une classe, tous ses membres seront privés, donc **inaccessibles**. Cela sera rarement utile.

```
class X
{
    private:
    ....
    public:
    ...
    private:
    ....
}
```

Remarque

- Si on rend publics tous les membres d'une classe, on obtient l'équivalent d'une structure. Ainsi, ces deux déclarations définissent **le même type** *point*

```
struct point
{
    int x;
    int y;
    void initialise (...);
    .....
};
```

```
class point
{
    public:
        int x;
        int y;
        void initialise (...);
        .....
};
```

Définitions

Encapsulation: mécanisme qui permet de regrouper (rassembler) les données et les méthodes dans une classe, en indiquant leurs droits d'accès (visibilité): private, public ou protected.

Encapsulation des données: permet de protéger les données (données private) → l'accès aux données se fait **uniquement** par les méthodes de la classe elle-même.

Programmation modulaire (procédurale) vs Programmation Orientée Objet

- Programmation modulaire/ procédurale: (exemple: [Langage C](#)) (basée sur les fonctions), avec une **séparation** (dissociation) entre les données et les fonctions. Exemple:
 - données: structure ETUDIANT (cin, nom,date_naiss, notes, moyenne)
 - Fonctions: saisir_etudiant(ETUDIANT), afficher_etudiant(ETUDIANT), etc
- Programmation orientée objet POO (exemple: [Langage C++](#)): **regroupement** des données et des méthodes. Elle se base sur 3 techniques: **Encapsulation**, **héritage** et **polymorphisme**. Exemple: classe point:
 - Données: x et y
 - Méthodes: initialiser, déplacer et afficher

The image shows a Visual Studio 2010 IDE with two windows. The main window displays a C++ source file named `main.cpp` with tabs for `main.cpp*`, `point.h`, and `point.cpp`. The code is in the `(Global Scope)` and `main()` context. It includes `<iostream>` and `"point.h"`, uses the `std` namespace, and defines a `main()` function. Inside `main()`, it prints a message, declares a `point` object `a`, initializes it with `(2,3)`, and prints its coordinates. Then, it prints another message, dynamically allocates a `point` object `*ad` with `new point()`, initializes it with `(5,6)`, and prints its coordinates. Finally, it calls `system("PAUSE");` to pause the execution.

```
#include <iostream>
#include "point.h"
using namespace std ;
void main()
{
    cout<<"\n creation d'un objet par declaration ";
    point a;
    // point a(); ==> erreur
    a.initialiser(2,3);
    a.afficher();
    cout<<"\n *****"<<endl;
    cout<<"\n creation d'un objet par allocation ";
    point *ad=new point;
    // ou bien    point *ad=new point();
    ad->initialiser(5,6);
    ad->afficher();
    system("PAUSE");
}
```

The Output window shows the execution results:

```
creation d'un objet par declaration
0019FA00
coordonnees: 2 3

*****

creation d'un objet par allocation
004348A0
coordonnees: 5 6
Appuyez sur une touche pour continuer...
```

Fonctions d'accès: accesseurs & mutateurs

- Permettent de retourner ou de modifier un champ (attribut) de la classe

Accesseurs (getter)

```
int getX();      // retourne la valeur de x  
int getY();      // retourne la valeur de y
```

Mutateurs (setter)

```
void setX(int);  // modifie la valeur de x  
void setY(int);  // modifie la valeur de y
```

#pragma once

class point

{

int x;

int y;

public:

void initialise (int, int) ;

void deplace (int, int) ;

void affiche () ;

int getX();

int getY();

void setX(int);

void setY(int);

};

Les setteurs (mutateurs)

Les getteurs (accesseurs)

#include <iostream>

#include "point.h"

using namespace std ;

void point::initialise (int abs, int ord) { ... }

void point::deplace (int dx, int dy) { ... }

void point::affiche () { ... }

int point::getX()

{ return x;

}

int point::getY()

{ return y;

}

void point::setX(int abs)

{ x=abs;

}

void point::setY(int ord)

{ y=ord;

}

```
main.cpp* point.h* x point.cpp
(Global Scope)
class point
{
    int x,y;
public:
    void initialiser(int,int);
    void deplacer(int,int);
    void afficher();
    // getteurs, accesseurs (déclarés inline)
    int point::getX() {return x;}
    int point::getY() {return y;}
    // setteurs, mutateurs (déclarés inline)
    void point::setX(int abs) {x=abs;}
    void point::setY(int ord) {y=ord;}
};
```

The image shows a Visual Studio code editor window with three tabs: `main.cpp`, `point.h`, and `point.cpp`. The `main.cpp` tab is active, showing the following code:

```
#include <iostream>
#include "point.h"
using namespace std ;

void main()
{
    point a;
    a.initialiser(2,3);
    cout<<"\n abscisse: "<<a.getX()<<"  ordonnee: "<<a.getY()<<endl;

    a.setX(99);
    a.setY(88);

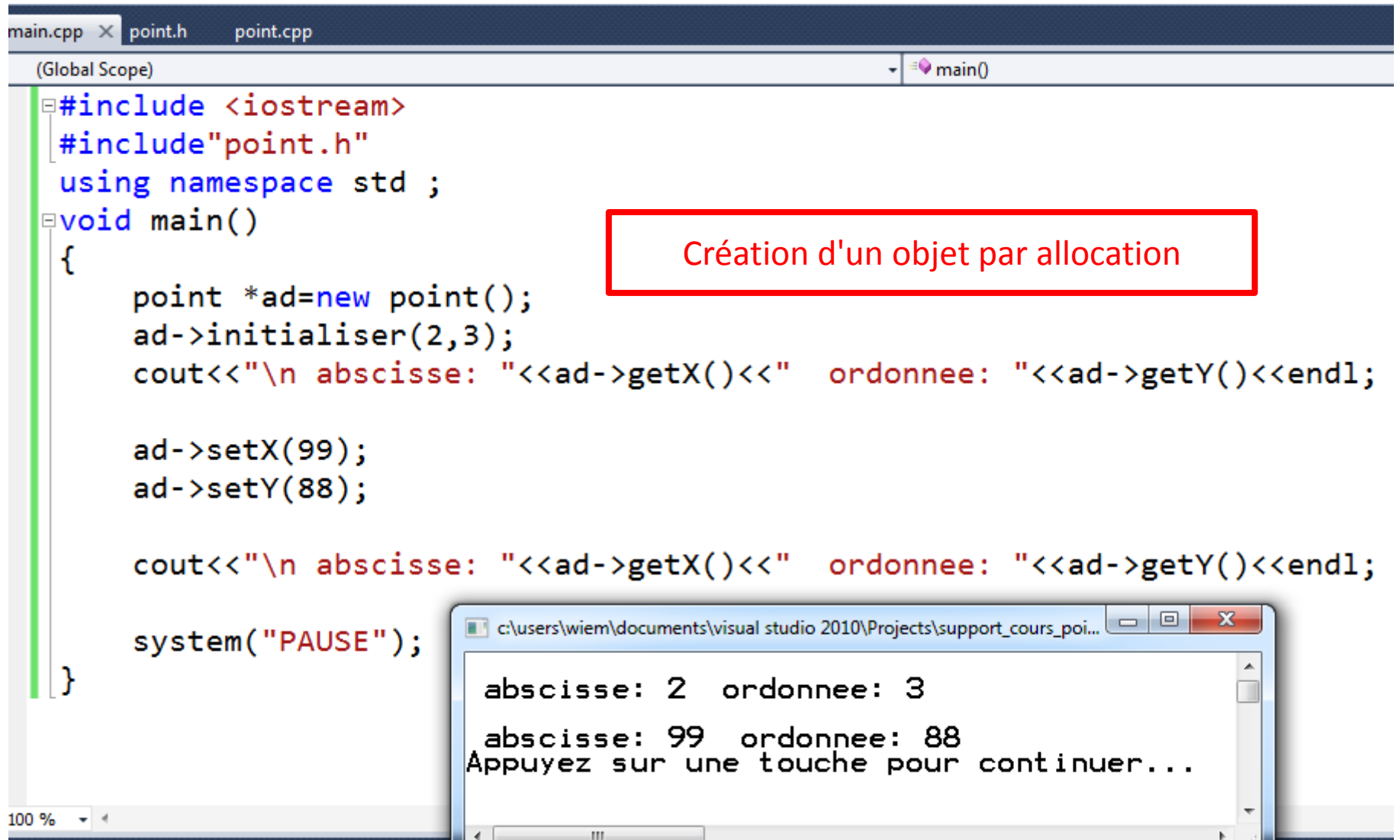
    cout<<"\n abscisse: "<<a.getX()<<"  ordonnee: "<<a.getY()<<endl;

    system("PAUSE");
}
```

A red rectangular box highlights the line `point a;` with the text "Création d'un objet par déclaration" (Creation of an object by declaration) written in red.

In the bottom right corner, a small window titled `c:\users\wiem\documents\visual studio 2010\Projects\support_cours_p...` displays the program's output:

```
abscisse: 2  ordonnee: 3
abscisse: 99  ordonnee: 88
Appuyez sur une touche pour continuer...
```



```
main.cpp x point.h point.cpp
(Global Scope) main()
#include <iostream>
#include "point.h"
using namespace std ;
void main()
{
    point *ad=new point();
    ad->initialiser(2,3);
    cout<<"\n abscisse: "<<ad->getX()<<" ordonnee: "<<ad->getY()<<endl;

    ad->setX(99);
    ad->setY(88);

    cout<<"\n abscisse: "<<ad->getX()<<" ordonnee: "<<ad->getY()<<endl;

    system("PAUSE");
}
```

Création d'un objet par allocation

c:\users\wiem\documents\visual studio 2010\Projects\support_cours_poi...
abscisse: 2 ordonnee: 3
abscisse: 99 ordonnee: 88
Appuyez sur une touche pour continuer...

plan

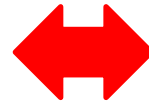
1. Les structures en C++
2. Notion de classe
- 3. Affectation d'objets**
4. Notion de constructeur et de destructeur
5. Surdéfinition des fonctions membres
6. Arguments par défaut des fonctions membres
7. Les membres données et fonctions statiques
8. Protection contre les inclusions multiples
9. Cas 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

Affectation d'objets

- En C, il est possible d'affecter à une variable structure la valeur d'une autre variable structure **de même type**.

```
struct point
{
    int x;
    int y;
};
struct point a,b;
```

a=b;



```
a.x=b.x;
a.y=b.y;
```

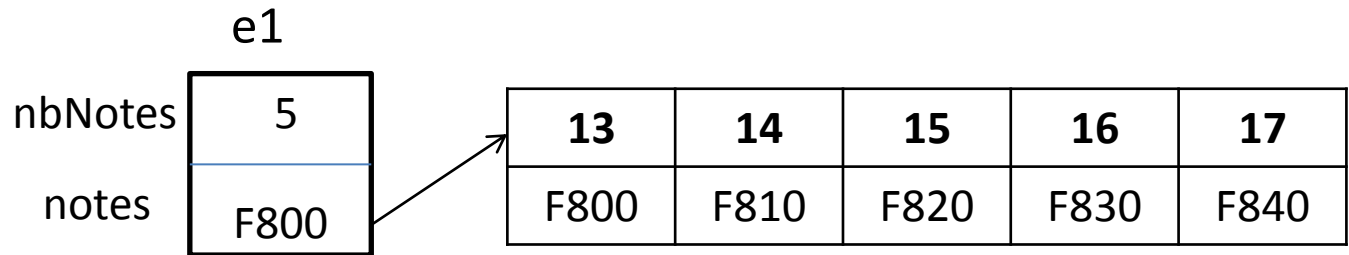
- a=b; recopie l'ensemble des valeurs des champs de b dans ceux de a.

Affectation d'objets

- En C++, cette possibilité d'**affectation globale** s'étend aux **objets de même type**.
- l'affectation $a = b$ est toujours légale, quel que soit le statut (**public ou privé**) des membres données.
- L'affectation entre deux objets est insuffisante **quand l'objet comporte des pointeurs sur des emplacements dynamiques**: la recopie en question ne concernera pas cette partie dynamique de l'objet.

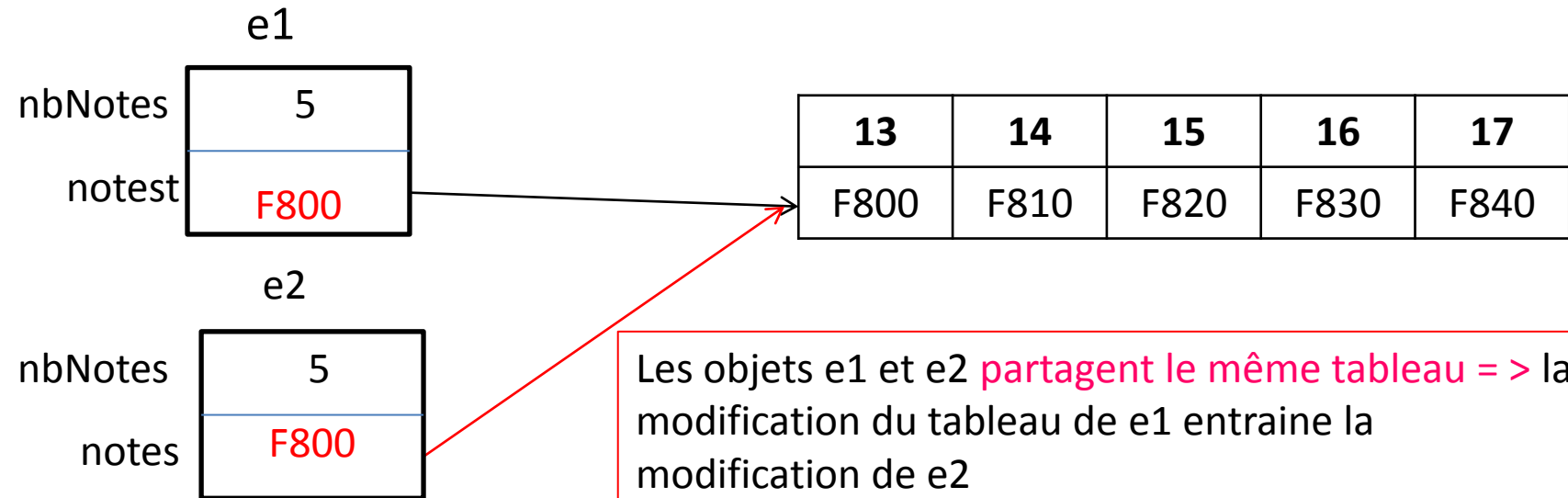
Problème: copie superficielle

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



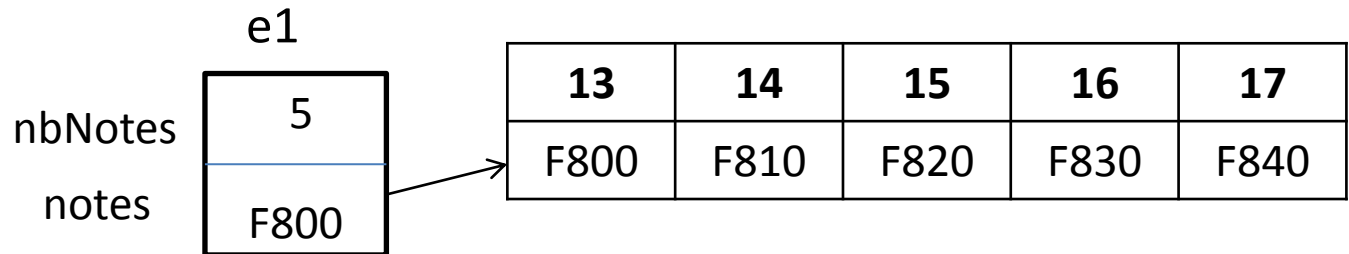
etudiant e1;
L'objet e1 contient une partie dynamique

e2=e1;

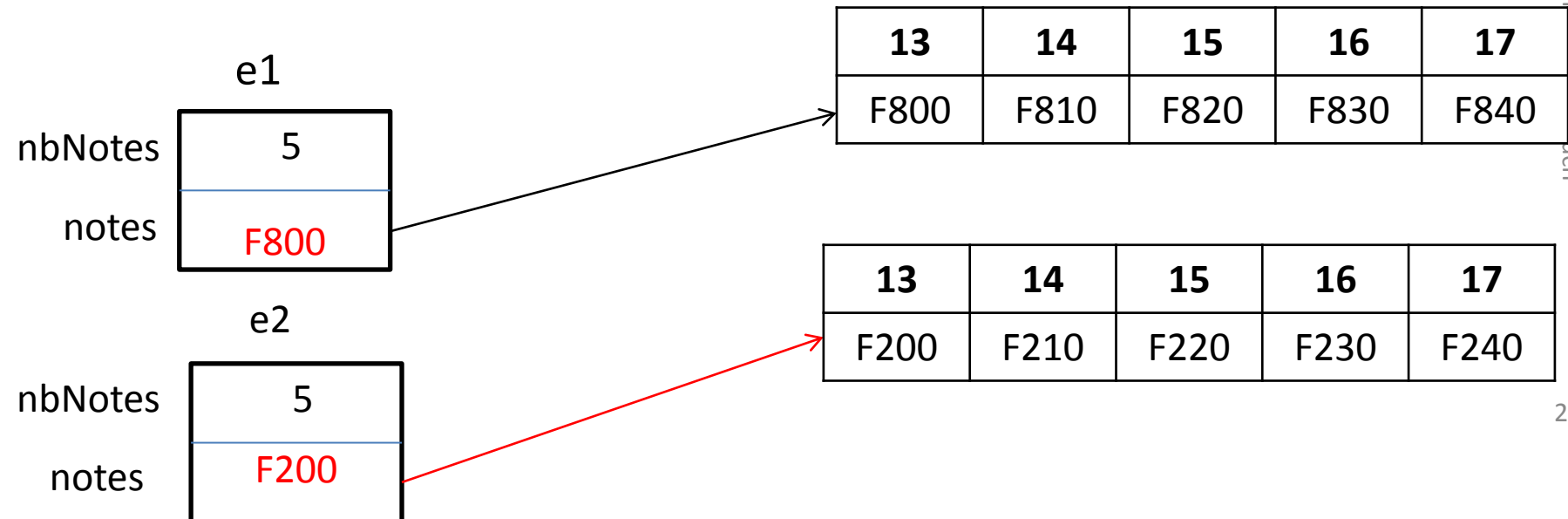


Solution: copie profonde

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



e2=e1



plan

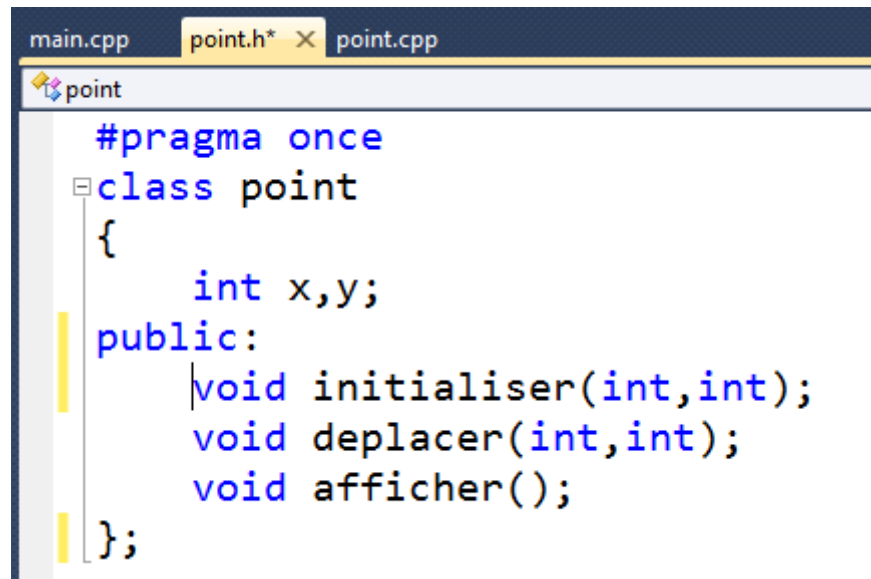
1. Les structures en C++
2. Notion de classe
3. Affectation d'objets
- 4. Notion de constructeur et de destructeur**
5. Surdéfinition des fonctions membres
6. Arguments par défaut des fonctions membres
7. Les membres données et fonctions statiques
8. Protection contre les inclusions multiples
9. Cas 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

constructeur & destructeur

- En général, il est nécessaire de faire appel à une fonction membre pour attribuer des valeurs aux données d'un objet. c'était la fonction *initialiser* pour le type *point*.
- C++ offre un mécanisme très performant : le **constructeur**. Il s'agit d'une **fonction membre** (définie comme les autres fonctions membres) qui sera appelée **automatiquement** à chaque création d'un objet (**par déclaration ou par new**).

constructeur & destructeur

- Un objet pourra aussi posséder **un destructeur**, c'est-à-dire une fonction membre appelée automatiquement au moment de la **destruction** de l'objet.
- Dans le cas des objets créés par déclaration, la destruction de l'objet a lieu lorsque l'on quitte le bloc ou la fonction où il a été déclaré.
- Par convention, le constructeur se reconnaît à ce qu'il porte le **même nom** que la classe.
- Quant au destructeur, il porte le **même nom** que la classe, précédé **d'un tilde (~)**.



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

Lorsqu'aucun constructeur n'est défini, il existe un constructeur par défaut

(Global Scope)

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

class point
{
private:
    int x;
    int y;

public:
    point(int,int); // constructeur
    void deplacer(int,int);
    void afficher();
    ~point(); // destructeur
};
```

Déclaration d'un constructeur à 2 arguments

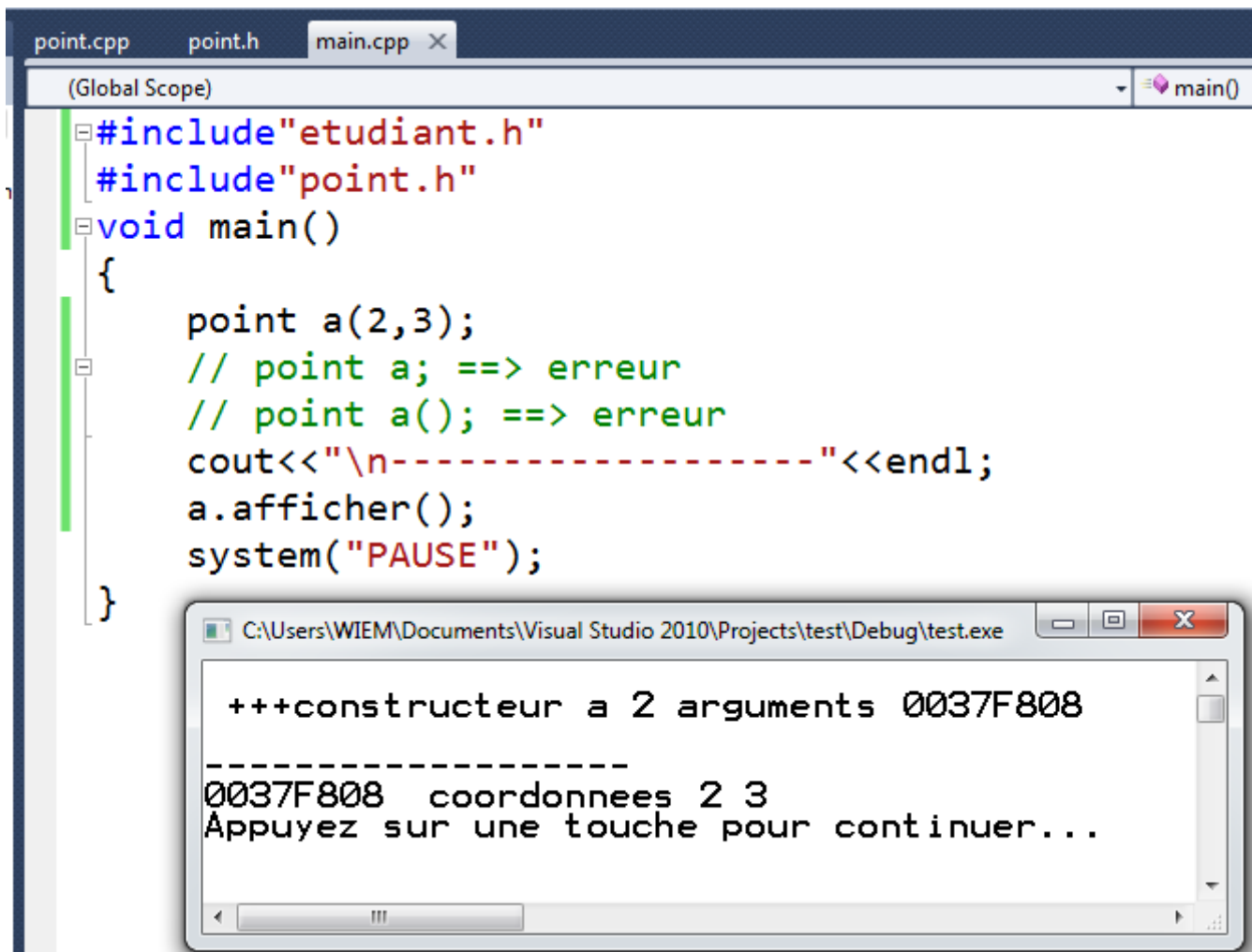
this: pour afficher l'adresse de l'objet

point

afficher()

```
point::point(int abs, int ord)
{
    cout<<"\n +++constructeur a 2 arguments "<<this<<endl;
    x=abs;
    y=ord;
}

point::~~point()
{
    cout<<"\n appel du destructeur "<<this<<endl;
}
```

The image shows a C++ IDE with three tabs: point.cpp, point.h, and main.cpp. The main.cpp tab is active, showing the following code:

```
#include "etudiant.h"
#include "point.h"
void main()
{
    point a(2,3);
    // point a; ==> erreur
    // point a(); ==> erreur
    cout<<"\n-----"<<endl;
    a.afficher();
    system("PAUSE");
}
```

Below the code editor, a debug console window is open, displaying the output of the program:

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe
+++constructeur a 2 arguments 0037F808
-----
0037F808  coordonnees 2 3
Appuyez sur une touche pour continuer...
```

point a; est correcte si:

1. Aucun constructeur n'est défini (appel du constructeur par défaut)
2. Ou bien, il existe un constructeur sans arguments

The image shows a Visual Studio 2010 IDE window with the file `main.cpp` open. The code defines a `point` struct and a `main` function. The `main` function creates a `point` object `q` with coordinates (2, 3) using `new`, prints its address and coordinates, and then deletes it using `delete`. The `Output` window at the bottom shows the execution results, including the memory address `007E4940` and the coordinates `2 3`.

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

#include "etudiant.h"
#include "point.h"
void main()
{
    // création et destruction d'un point
    // par allocation
    point *q=new point(2,3);
    cout<<"\n-----"<<endl;
    q->afficher();
    cout<<"\n-----"<<endl;
    delete q;
    cout<<endl;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test.exe

```
+++constructeur a 2 arguments 007E4940
-----
007E4940  coordonnees 2 3
-----

appel du destructeur 007E4940
Appuyez sur une touche pour continuer...
```

constructeur

- A partir du moment où un constructeur est défini, il doit pouvoir être appelé (automatiquement) lors de la création de l'objet.
- Ici, le constructeur a besoin de deux arguments. Ceux-ci doivent obligatoirement être fournis.

```
point a (1,3) ;  
point *q=new point(5,6);
```

- le constructeur peut être surdéfini (surchargé) ou posséder des arguments par défaut.

remarque

- La fonction initialiser permet d'initialiser les valeurs d'un objet.
- Le constructeur permet aussi d'initialiser les valeurs d'un objet.
- La fonction initialiser est appelée **explicitement** par l'objet:
a.initialiser(2,3);
- Le constructeur est appelé **automatiquement** après la création de l'objet (càd après la réservation de son espace mémoire)

En résumé

- **Un constructeur**

- Fonction membre qui porte le même nom que la classe
- Possède 0 ou n arguments; sans type de retour
- Permet d'initialiser les données d'un objet
- Appelé automatiquement après la création d'un objet.

- **Un destructeur**

- Fonction membre qui porte le même nom que la classe, précédée par tilde(~)
- Sans arguments; sans type de retour
- Permet de libérer les espaces mémoires dynamiques associés à l'objet
- Appelé automatiquement avant la destruction d'un objet.

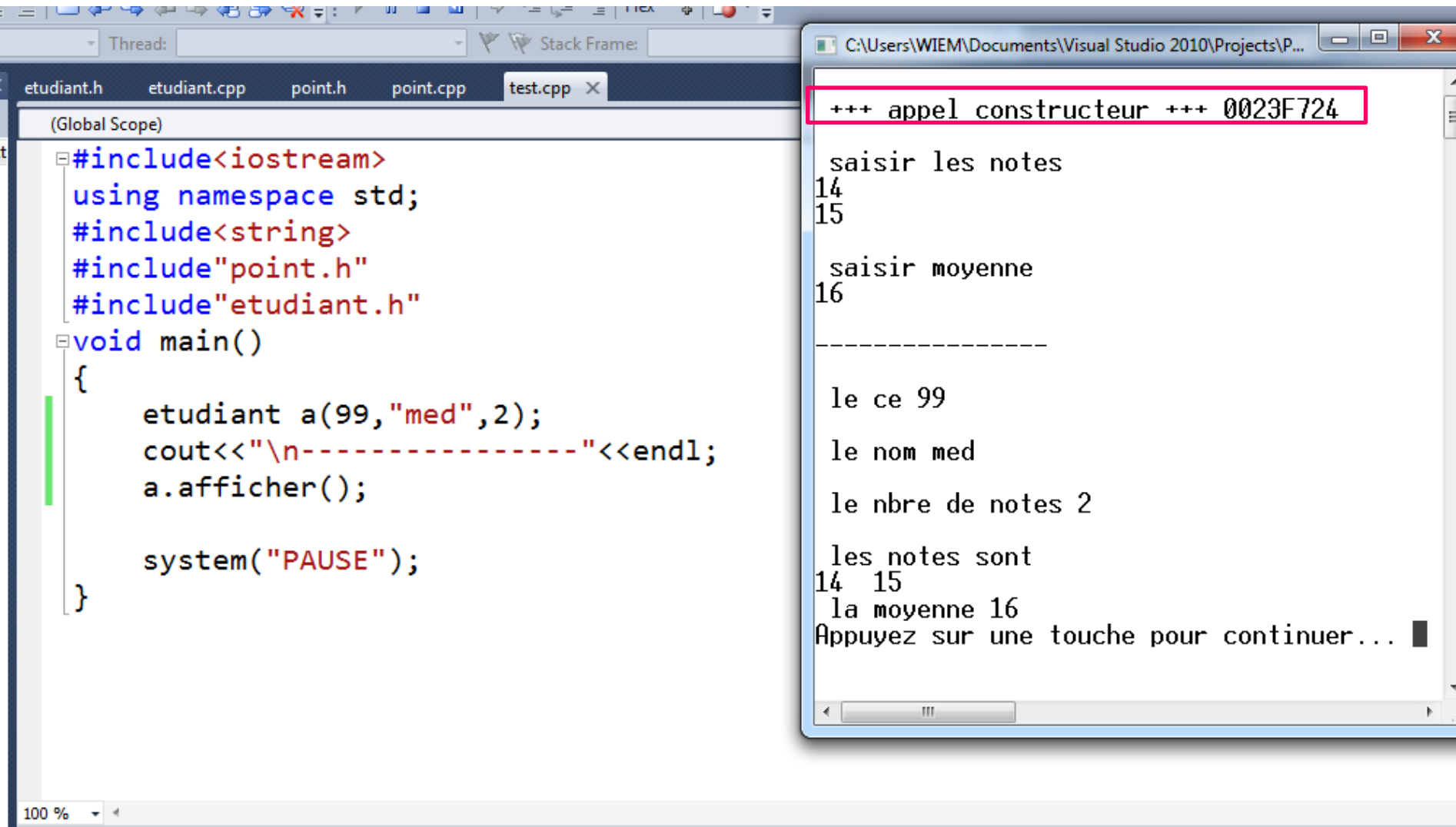
```
#pragma once
#include<iostream>
using namespace std;
#include<string>
class etudiant
{
    int ce;
    string nom;
    int nbNotes;
    int*notes;
    float moyenne;
public:
    etudiant(int,string,int);
    ~etudiant(void);
    void afficher();
    // getteurs setteurs
};
```

```
etudiant::etudiant(int c, string n, int nb)
{
    cout<<"\n +++ appel constructeur +++ "<<this<<endl;
    ce=c;
    nom=n;
    nbNotes=nb;
    notes=new int[nbNotes];
    cout<<"\n saisir les notes "<<endl;
    for(int i=0; i<nbNotes; i++)
        cin>>*(notes+i);
    cout<<"\n saisir moyenne "<<endl;
    cin>>moyenne;
}
```

```
etudiant::~~etudiant(void)
{
    cout<<"\n --- appel destructeur --- "<<this<<endl;
    delete[]notes;
}
```

```
void etudiant::afficher()
{
    cout<<"\n le ce "<<ce<<endl;
    cout<<"\n le nom "<<nom<<endl;
    cout<<"\n le nbre de notes "<<nbNotes<<endl;
    cout<<"\n les notes sont "<<endl;
    for(int i=0; i<nbNotes; i++)
        cout<<*(notes+i)<<" ";
    cout<<"\n la moyenne "<<moyenne<<endl;
}
```

Exécution 1: etudiant par déclaration



The image shows a screenshot of the Visual Studio 2010 IDE. The left pane displays the source code for a C++ program. The right pane shows the output of the program's execution.

Source Code (test.cpp):

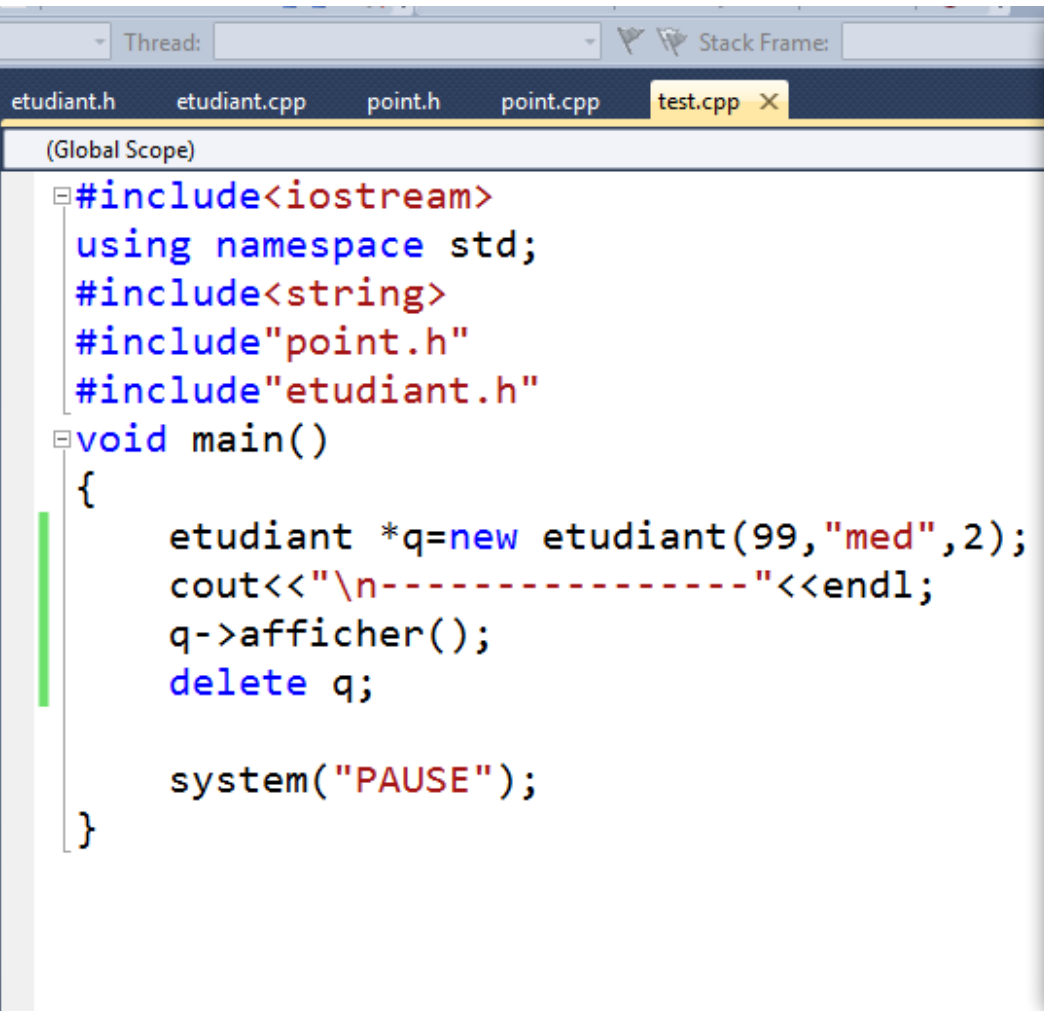
```
#include<iostream>
using namespace std;
#include<string>
#include"point.h"
#include"etudiant.h"
void main()
{
    etudiant a(99,"med",2);
    cout<<"\n-----"<<endl;
    a.afficher();

    system("PAUSE");
}
```

Output Window:

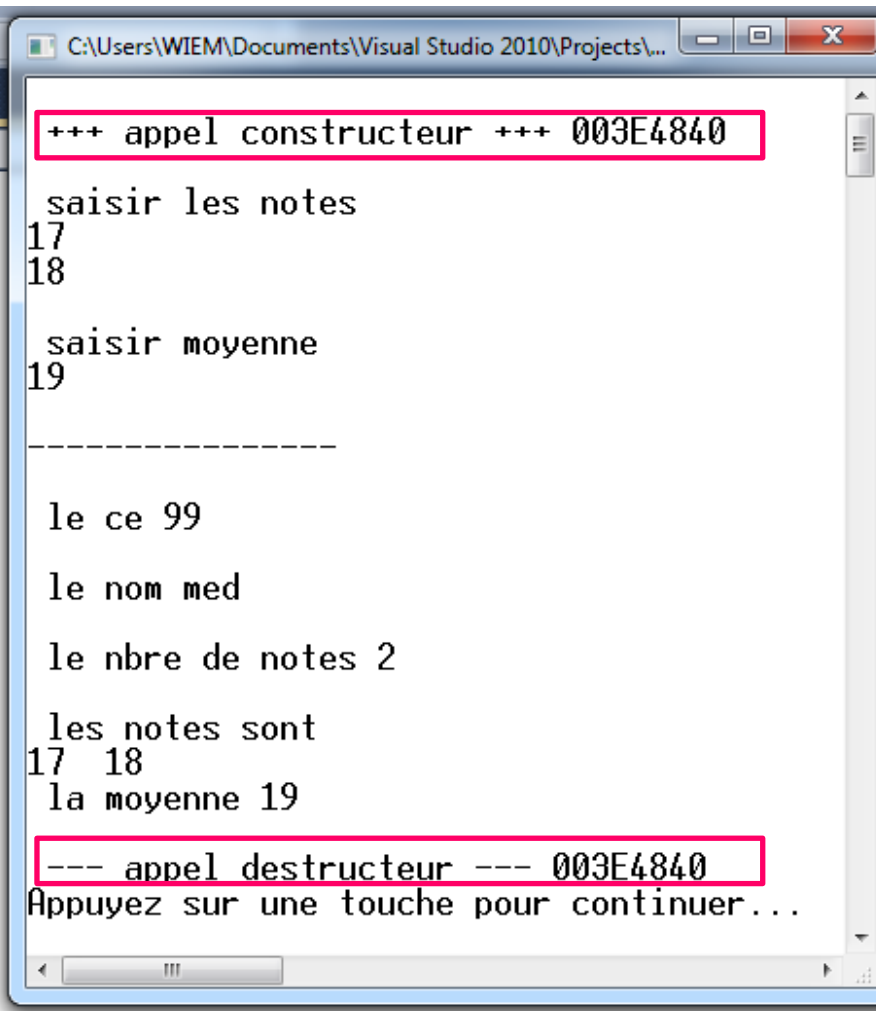
```
+++ appel constructeur +++ 0023F724
saisir les notes
14
15
saisir moyenne
16
-----
le ce 99
le nom med
le nbre de notes 2
les notes sont
14 15
la moyenne 16
Appuyez sur une touche pour continuer...
```

Exécution 2: etudiant par allocation



```
(Global Scope)
#include<iostream>
using namespace std;
#include<string>
#include"point.h"
#include"etudiant.h"
void main()
{
    etudiant *q=new etudiant(99,"med",2);
    cout<<"\n-----"<<endl;
    q->afficher();
    delete q;

    system("PAUSE");
}
```



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\...
+++ appel constructeur +++ 003E4840
saisir les notes
17
18
saisir moyenne
19
-----
le ce 99
le nom med
le nbre de notes 2
les notes sont
17 18
la moyenne 19
--- appel destructeur --- 003E4840
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

Surdéfinition des fonctions membres

- C++ autorise à surdéfinir les **fonctions ordinaires** (**fonctions indépendantes**), càd n'appartenant à aucune classe.
- Cette possibilité s'applique également aux **fonctions membres** d'une classe, y compris au **constructeur** (mais pas au destructeur puisqu'il ne possède pas d'arguments).

```
point.cpp  point.h*  main.cpp
point
#pragma once
#include<iostream>
using namespace std;
#include<string>
class point
{
private:
    int x;
    int y;
public:
    // surdéfinition du constructeur
    point();
    point(int);
    point(int,int);
    void déplacer(int,int);
    // surdéfinition de la fonction afficher
    void afficher();
    void afficher(string);
    ~point();
};
```

```
point.cpp* x point.h* main.cpp
point
#include "point.h"
point::point()
{
    cout<<"\n +++constructeur a 0 arguments "<<this<<endl;
    x=99;
    y=88;
}
point::point(int v)
{
    cout<<"\n +++constructeur a 1 argument "<<this<<endl;
    x=v;
    y=v;
}
point::point(int abs, int ord)
{
    cout<<"\n +++constructeur a 2 arguments "<<this<<endl;
    x=abs;
    y=ord;
}
```

Mme Wiem

Appel de la fonction afficher.
En effet, une fonction
membre peut toujours en
appeler une autre (qu'elle
soit publique ou non)

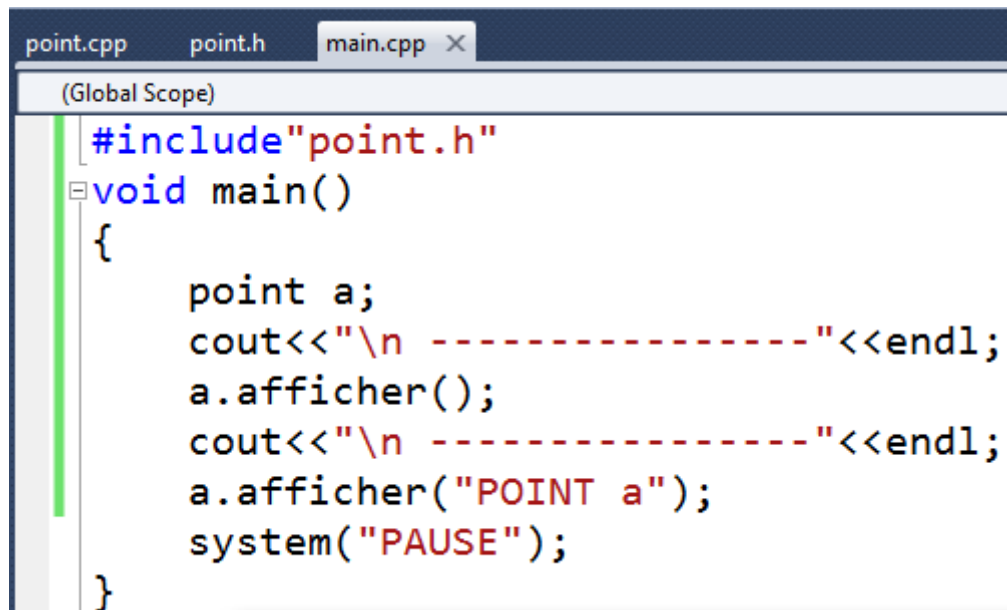
```
point.cpp x point.h main.cpp
point
void point::afficher()
{
    cout<<this<<" coordonnees "<<x<<" "<<y<<endl;
}
void point::afficher(string msg)
{
    cout<<msg<<" ";
    afficher();
}
```

```
point.cpp  point.h  main.cpp* X
(Global Scope)
#include "point.h"
void main()
{
    point a; // constr à 0 arg
    a.afficher(); // x:99, y:88
    cout<<"\n -----"<<endl;
    point b(5); // constr à 1 arg
    b.afficher(); // x:5, y:5
    cout<<"\n -----"<<endl;
    point c(2,3); // constr à 2 arg
    c.afficher(); // x:2, y:3
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\te...
+++constructeur a 0 arguments 004CF71C
004CF71C  coordonnees 99 88
-----

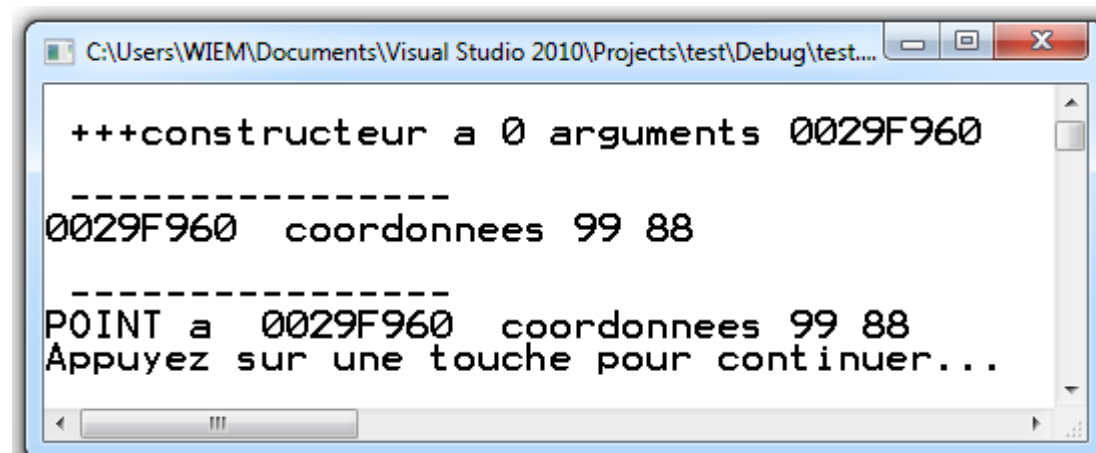
+++constructeur a 1 argument 004CF70C
004CF70C  coordonnees 5 5
-----

+++constructeur a 2 arguments 004CF6FC
004CF6FC  coordonnees 2 3
Appuyez sur une touche pour continuer...
```



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

#include "point.h"
void main()
{
    point a;
    cout<<"\n -----"<<endl;
    a.afficher();
    cout<<"\n -----"<<endl;
    a.afficher("POINT a");
    system("PAUSE");
}
```



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

+++constructeur a 0 arguments 0029F960

-----
0029F960  coordonnees 99 88

-----
POINT a 0029F960 coordonnees 99 88
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

Arguments par défaut des fonctions membres

- Comme les fonctions ordinaires (indépendantes), les fonctions membres peuvent disposer d'arguments par défaut.
- En utilisant les possibilités d'arguments par défaut, il est souvent possible de **diminuer** le nombre de fonctions surdéfinies.


```
point.cpp* point.h* main.cpp
(Global Scope)
#pragma once
#include<iostream>
using namespace std;
#include<string>
class point
{
private:
    int x;
    int y;
public:
    point(int =99,int =88);
    void deplacer(int,int);
    void afficher(string = "");
    ~point();
};
```

```
point.cpp x point.h main.cpp
point
#include "point.h"
point::point(int abs,int ord)
{
    cout<<"\n +++constructeur "<<this<<endl;
    x=abs;
    y=ord;
}
void point::afficher(string msg)
{
    cout<<msg<<" ";
    cout<<this<<" coordonnees "<<x<<" "<<y<<endl;
}
```

- point est un constructeur à 2 arguments qui peut être appelé avec 0 ou 1 ou 2 arguments
- afficher est une fonction à 1 argument qui peut être appelée avec 0 ou 1 argument.

Le constructeur
peut être défini
inline

```
class point
{
private:
    int x;
    int y;
public: // constructeur inline
    point(int abs=99,int ord=88)
    {
        cout<<"\n +++constructeur "<<this<<endl;
        x=abs;
        y=ord;
    }
    void deplacer(int,int);
    void afficher(string = "");
    ~point();
};
```

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

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\te...
+++constructeur 0042FBC4
-----
+++constructeur 0042FBB4
-----
+++constructeur 0042FBA4
Appuyez sur une touche pour continuer...
```

```
point.cpp  point.h  main.cpp X
(Global Scope)
#include "etudiant.h"
#include "point.h"
void main()
{
    point a;
    cout<<"\n -----"
    a.afficher();
    cout<<"\n -----"
    a.afficher("POINT a");
    cout<<endl;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test\Debug\test....
+++constructeur 003CFAE4
-----
003CFAE4  coordonnees 99 88
-----
POINT a 003CFAE4  coordonnees 99 88
Appuyez sur une touche pour continuer...
```