

# Programmation Orientée Objet

Programmation des classes et objets

**Pr Imane DAOUDI**

**2016-2017**

# Plan

1. Introduction à la programmation orientée objet
2. Généralités
3. Opérateurs sur tableaux, pointeurs et objets
4. Les fonctions
5. Programmation des classes et objets
6. Appels et surcharges
7. L' héritage
8. Polymorphisme

# 5. Programmation des classes et objets

## ➤ La classe en C++.

**La classe en C++** est la généralisation de la notion **de la structure du C.**

Elle comporte : **-Des champs** ( les données )

**-Des fonctions** ( les méthodes agissent sur les données )

Le mot clé « **class** » permet de définir et déclarer la classe.

La classe doit être déclarée **avant** d'être utilisée.

La déclaration d'une classe précise :quels sont ses membres qui peuvent être :

- **publics** : à l'aide du mot clé « **public** » accessibles à l'utilisateur de la classe.
- **privés** : mot clé « **private** » inaccessibles en dehors de la classe.

Remarque : ***Par défaut les membres sont privés.***

# 5. Programmation des classes et objets

## ➤ La classe en C++

- Une **classe** est une structure dans laquelle seulement certains membres et/ou certaines méthodes sont accessibles depuis l'extérieur, les autres étant seulement accessibles à l'intérieur.
- Une classe permet donc d'encapsuler des données en les déclarant privées (mot clef **private**) !
- La déclaration d'une classe est voisine de celle d'une structure :
  - il faut remplacer le mot clef **struct** par **class** ;
  - il faut préciser quels membres (données ou fonctions) sont publics ou privés à l'aide des mots clés **public** et **private**. Si tous les membres sont publics on obtient l'équivalent d'une structure !

# 5. Programmation des classes et objets

## ➤ La classe en C++

```
class complex
{
    private :           // données privées (encapsulées)
                        float im, re;

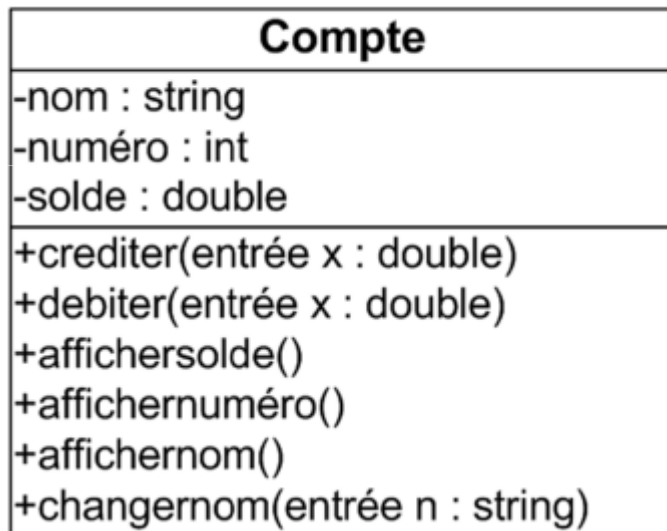
    public :
                        // méthodes publiques
                        void initialise() ;
                        void modifie(float, float) ;
                        void affiche() ;
};           // le ; est indispensable
```

- Les méthodes d'une classe ont accès à toutes les données de la classes quelles soient **publiques** ou **privées** !

# 5. Programmation des classes et objets

## ➤ Déclaration d'une classe en C++ :

- Exemple : déclarons une classe « Compte » correspondant à ce diagramme UML



```
class Compte
{
    private :
        string nom;
        int numero;
        double solde;
    public :
        void crediter (double x);
        void debiter (double x);
        void affichersolde (void);
        void affichernumero (void);
        void affichernom (void);
        void changernom (string n);
};
```

## 5. Programmation des classes et objets

### ➤ Définition des méthodes en C++

Consiste à donner les définitions des fonctions membres suivant la syntaxe :

```
type_retour  nom_classe  :: nom_fc (liste des paramètres)
                {
                }
```

Remarque :

*La définition* d'une fonction membre peut se faire à l'intérieur d'une déclaration de classe ou à l'extérieur à l'aide de l'opérateur ::

**« :: » est l'opérateur de résolution de portée.**

# 5. Programmation des classes et objets

## ➤ Définition des méthodes en C++

Exemple 1 : définition des fonctions membres à l'extérieur de la classe

```
class point
{
    private :
        int x ;
        int y;
    public :
        void initialisation(int, int) ;
        void deplace(int, int) ;
        void affiche( ) ;
};
void point :: initialisation(int abs, int ord)
{
    x = abs ;
    y = ord ;
}
// x et y représentent implicitement les membres d'un objet de class point
```



# 5. Programmation des classes et objets

## ➤ Définition des méthodes en C++

Exemple 2 : définition des fonctions membres à l'intérieur de la classe

**class** article

{

**int** num ;

**char** \*nom ;

**float** prixHT ;

**int** qte ;

**public :**

**float** prixTTC() { return prixHT\*1.19 ; } // qui calcule le prix TTC

**void** ajouter(int q) { qte = qte + q ; } // augmentation du stock

**void** retirer( int q) { qte = qte - q ; } // diminution du stock

};

## 5. Programmation des classes et objets

### ➤ Utilisation des méthodes en C++

#### Identiques au cas d'une structure

*La syntaxe pour déclarer un objet de type classe définie est :*

```
nom_classe liste_des_noms_objets ;
```

*Exemple : point a, b ; article c, d ;*

On accède à un membre public ( donnée ou fonction) à l'aide de l'opérateur « . » :

Exemple : **c.ajouter(50) ; a.initialise(5,2) ; x = d.prixTTC() ;**

- Ces exemples représentent des appels de fonctions membres des classes point ou article.


## 5. Programmation des classes et objets

- Découpage du code en modules:
- En C++ un module sera un regroupement logique et cohérent de classes.
- Pour chaque module on va créer deux types de fichiers :
- Un « **header** » c'est à dire un fichier « **.h** » qui contiendra les déclarations des classes du module.
- Un **fichier source** « **.cpp** » qui contiendra leurs implémentations (définitions).

## 5. Programmation des classes et objets

### ➤ Découpage du code en modules: exemple

Notre « main » devra nécessairement inclure notre nouveau fichier « .h » :



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

int main()
{
    ...
}
```

## 5. Programmation des classes et objets

### ➤ Découpage du code en modules: exemple

Le fichier « .h » :

```
#ifndef MACLASSE_H_INCLUDED
#define MACLASSE_H_INCLUDED
#include <string>
class Compte
{ private :
    std::string nom;
    int numero;
    double solde;
public :
    void crediter (double x);
    void debiter (double x);
    void affichersolde (void);
    void affichernumero (void);
    void affichernom (void);
    void changernom (std::string n);};
#endif // MACLASSE_H_INCLUDED
```

## 5. Programmation des classes et objets

### ➤ Découpage du code en modules: exemple

**Le fichier « .cpp » :**

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

void Compte::crediter (double x)
{ solde += x; }

void Compte::debiter (double x)
{ solde -= x; }
```

## 5. Programmation des classes et objets

### ➤ Découpage du code en modules: exemple

#### Le fichier « .cpp » (suite)

```
void Compte::affichersolde(void)
{ cout << "solde : " << solde << endl; }
```

```
void Compte::affichernumero(void)
{ cout << "numero de compte : " << numero << endl; }
```

```
void Compte::affichernom(void)
{ cout << "nom du compte : " << nom << endl; }
```

```
void Compte::changernom (string n)
{ nom=n; }
```

# 5. Programmation des classes et objets

## ➤ Découpage du code en modules: exemple

### ➤ **Remarque fondamentale :**

- Dans un fichier « .cpp », quand vous implémentez les méthodes d'une classe définie dans un fichier « .h », vous devez rajouter devant le nom des méthodes le préfixe :


***NomDeLaClasse::***

### ➤ **Une autre remarque fondamentale :**

- Les méthodes d'une classe disposent d'un accès direct aux attributs de cette même classe, c'est pourquoi il n'est pas nécessaire de passer ces derniers en paramètres pour qu'une méthode puisse les manipuler.
- **Exemple :**

```
void Compte::debiter (double x)  
{ solde -= x; }
```

Solde est un  
attribut de  
la classe





## 5. Programmation des classes et objets

### ➤ Découpage du code en modules: exemple

#### ➤ **Encore une remarque fondamentale**

- Si une méthode n'est pas sensé modifier la valeur des attributs de la classe, on pourra lui adjoindre le suffixe « **const** ». C'est une précaution contre d'éventuelles erreurs de code, car dans le cas où'on aurait une tentative de modification d'un attribut le compilateur le signalerait.
- **Exemple :**

```
void affichersolde (void) const;
```

#### ➤ **Une dernière remarque fondamentale :**

- Il est interdit d'initialiser les attributs lors de la déclaration d'une classe.

## 5. Programmation des classes et objets

### ➤ Constructeurs

Relativement à un objet de classe il doit exister une opération caractéristique, dont l'utilité est de fournir une première initialisation à cet objet.

⇒ La meilleure approche est de définir une fonction dont le rôle systématique est d'initialiser l'objet.

⇒ Une telle opération est appelée « constructeur »

*En C++, Un constructeur est une « fonction membre » qui porte le même nom que sa classe.*

# 5. Programmation des classes et objets

## ➤ Constructeurs

- Le constructeur **alloue de la mémoire à un objet** et **l'initialise**.
- Le constructeur ne doit être précédé ni d'un type donné ni de « void ».

**class** article

```
{ .....  
    article(int, char *, float, int);    // constructeur de la classe article  
    .....  
};
```

- Toutes les instances de cette classe sont initialisées par cette fonction membre.

# 5. Programmation des classes et objets

## ➤ Constructeurs

**Exemple 1 :**

```
article a = article(102, "chemise",98.4, 23) ;
```

le constructeur est appelé dès la déclaration pour fournir les valeurs initiales

⇒ **a** est initialisé par affectation d'une instance créée par le constructeur.

**Exemple 2 :**

```
article a(102, "chemise",98.4, 23) ;
```

C'est la forme abrégée du constructeur , car elle fait appel automatiquement au constructeur.

# 5. Programmation des classes et objets

## ➤ Constructeurs

Exemple : `article b = a ;`  
`article c ;`

- L'objet (ou instance) « **b** » est initialisé à partir de l'objet « a » supposé existant.
- L'écriture pour la création de l'objet « **c** » est incorrecte

⇒ Un objet ne peut être déclaré sans être initialisé : affectation créée par un constructeur ou un objet déjà existant.

Exemple 1 : Dans le cas d'un pointeur sur un objet

```
article *pa = new article(102,"chemise",98.4, 23) ;
```

Ou

```
article *pa ;  
pa = new article(102,"chemise",98.4, 23) ;
```

new sert à allouer de l'espace mémoire à un pointeur.

# 5. Programmation des classes et objets

## ➤ Constructeurs

### Propriétés d'un constructeur

- Le **nom d'un constructeur** est toujours le **même** que celui de **la classe**.
- Les constructeurs **ne retournent aucune valeur** (même pas void).
- Conformément au principe de surcharge des fonctions, **une classe** peut contenir **plusieurs constructeurs**.
- Les paramètres des constructeurs peuvent avoir des valeurs par défaut (comme toute fonction en C++).

# 5. Programmation des classes et objets

## ➤ Constructeurs

### Remarques

1. Possibilité de définition d'un constructeur à l'intérieur ou à l'extérieur d'une classe
2. Quand le constructeur n'a pas de paramètres , on l'appelle « **constructeur par défaut** ».
3. Il est souvent intéressant **d'avoir plus d'un** constructeur pour initialiser des objets d'une classe.
4. Tous les constructeurs **ont le même nom**. Ils seront différenciés par le nombre et le type des arguments.
5. Suivant l'appel , on sait que c'est tel ou tel constructeur qui est appelé( voir exemple ).

# 5. Programmation des classes et objets

## ➤ Constructeurs

**Exemple** : soit une classe « **date** ». Voici l'appel de différents constructeurs de cette classe :

```
date( ) ; // date par défaut
```

```
date( int, int, int ) ; // donner jour , mois , année
```

```
date( char * ) ; // sous forme "12/04/95"
```

```
date( int , char * , int ) ; // sous forme 12 Avril 95
```



# 5. Programmation des classes et objets

## ➤ Constructeurs

### Exemple de constructeur pour la classe compte:

Signature dans le « .h » :

```
Compte(double x,int i,std::string n);
```

Implémentation dans le « .cpp » :

```
Compte::Compte(double x,int i,string n)  
{  
    solde = x;  
    numero = i;  
    nom = n;  
}
```

## 5. Programmation des classes et objets

### ➤ Constructeurs

#### Exemple d'utilisation:

- maintenant déclarons un compte dans le « main » en faisant appel à ce constructeur :

```
Compte moncompte(100000,666,"Laurent");
```

ou

```
Compte moncompte = Compte(100000,666,"Laurent");
```

```
solde : 100000
numero de compte : 666
nom du compte : Laurent

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

# 5. Programmation des classes et objets

## ➤ Constructeurs

### Exemple d'utilisation:

- Autre façon d'implémenter le même constructeur : **avec une liste d'initialisation**

```
Compte::Compte(double x,int i,string n) : solde(x),numero(i), nom(n)
{
}
```

Cela nécessite que les attributs prennent directement les valeurs passées en paramètres au constructeur.

# 5. Programmation des classes et objets

## ➤ Constructeurs

- **Autre exemple de constructeur** : avec une liste d'initialisation pour certains attributs et une initialisation dans le "corps" du constructeur pour d'autres

```
Compte::Compte(double x,int i,string n) : numero(i), nom(n)
{
    solde = x - 100;
}
```

- **Constructeur par défaut** : il s'agit d'un constructeur que l'on peut utiliser sans passer de paramètres. C'est donc un constructeur ne possédant pas de paramètres ou dont les paramètres possèdent tous une valeur par défaut.
- A partir de maintenant, **toutes nos classes contiendront un constructeur par défaut**, cela permettra que des déclarations du type aient un minimum de sens.

```
Compte moncompte;
```

# 5. Programmation des classes et objets

## ➤ Constructeurs

- **Exemple de constructeur par défaut** : modification du constructeur précédent avec valeurs des paramètres par défaut :
- Dans le « .h » :

```
Compte(double x=0,int i=0,std::string n="NoName");
```

- Dans le « .cpp » pas de modifications :

```
Compte::Compte(double x,int i,string n)  
{ solde = x;  
  numero = i;  
  nom = n; }
```

# 5. Programmation des classes et objets

## ➤ Constructeurs

- **Autre exemple de constructeur par défaut** : constructeur sans paramètres.
- Dans le « .h » :

```
Compte ();
```

- Dans le « .cpp » :

```
Compte::Compte()  
{ solde = 0;  
  numero = 0;  
  nom = "NoName"; }
```

Une classe ne peut contenir au plus qu'un constructeur par défaut. Par contre, conformément au principe de surcharge des fonctions, une classe peut contenir plusieurs constructeurs de signatures différentes.

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **L'accès aux attributs et méthodes publiques d'un objet se fait avec l'opérateur « . » :**
  - NomDeL'objet.NomDeL'attribut
  - NomDeL'objet.NomDeLaMéthode(...)
- **L'accès aux attributs et méthodes privés d'un objet est interdit.**

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Exemple** : l'affichage déjà rencontré

```
solde : 100000  
numero de compte : 666  
nom du compte : Laurent
```

- correspond au code :

```
int main()  
{ Compte moncompte(100000,666,"Laurent");  
  moncompte.affichersolde();  
  moncompte.affichernumero();  
  moncompte.affichernom(); }
```

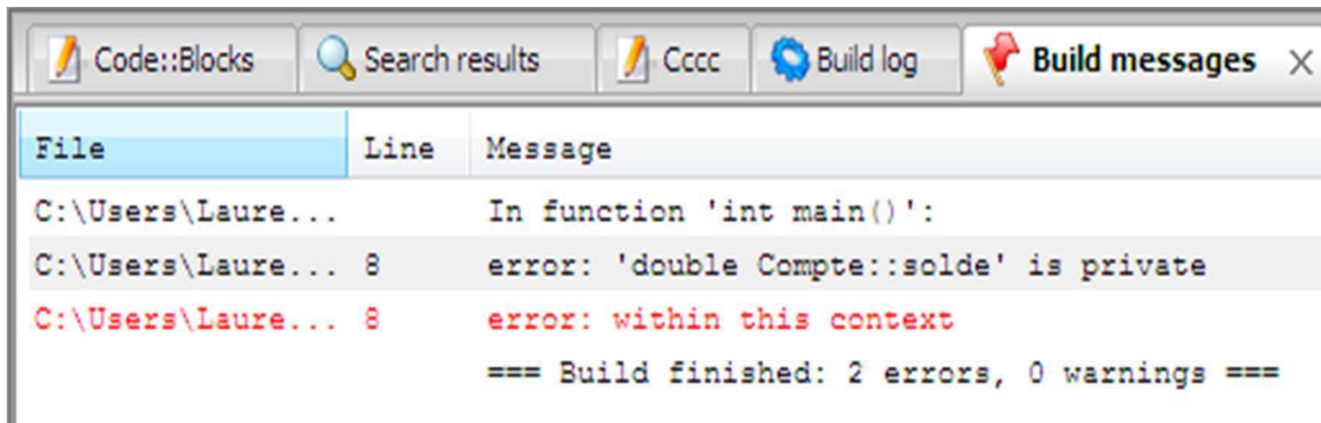


## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Exemple** : accès interdit à un attribut privé

```
int main()
{ Compte moncompte(100000,666,"Laurent");
  moncompte.solde = 100000; }
```



File	Line	Message
C:\Users\Laure...		In function 'int main()':
C:\Users\Laure...	8	error: 'double Compte::solde' is private
C:\Users\Laure...	8	error: within this context
=== Build finished: 2 errors, 0 warnings ===		

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Comment alors accéder aux attributs privés ?**
- Via des méthodes de la classe qui elles, rappelons le, ont accès à tous les attributs mêmes privés :
  - en lecture, ce sera des « getter » comme la méthode « affichersolde » de l'exemple précédent.
  - en écriture, ce sera des « setter » comme par exemple la méthode « changernom ».

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- Il peut arriver qu'une **fonction extérieure à la classe** ait besoin de manipuler **les attributs privés** de celle-ci.
- Pour cela, on la déclare amie de la classe, en mettant sa signature précédée du mot clé « friend » dans la déclaration de la classe.
- Elle aura alors accès sans restriction à tous les membres de la classe.

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Exemple** : on déclare comme amie de la classe « Compte » la fonction « impôts », et ce dans la déclaration de la classe.

```
friend void impots (Compte &c, double tax);
```

On implémente ensuite (en dehors de la classe puisque c'est une fonction extérieure) notre fonction :

```
void impots (Compte &c, double tax)
{ c.solde-= tax;
  cout << "prelevement automatique de " << tax <<
  " euros pour impots impayes" << endl; }
```

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Exemple** : cette fonction a alors accès aux attributs privés de la classe.

```
int main()
{ Compte compte(100000,666,"Laurent");
  compte.afficheSolde();
  impots(compte,10000);
  compte.afficheSolde(); }
```

```
solde : 100000
prelevement automatique de 10000 euros pour impots impayes
solde : 90000
```

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Accès aux attributs et méthodes de classe**
- S'ils sont privés l'accès est interdit.
- S'ils sont publics, l'accès se fait à l'aide de **l'opérateur « :: » de résolution de portée** :
- `NomDeLaClasse::NomDeL'attribut`
- `NomDeLaClasse::NomDeLaMéthode( ... )`.

## 5. Programmation des classes et objets

### ➤ Constructeurs: Accès aux membres des classes

- **Exemple** : dans le cas de notre classe « Compte », pour afficher le nombre de comptes créés, on utilisera la commande

```
Compte::affichernbcomptes();
```

- Cela affichera « 0 », et c'est normal car c'est avec cette valeur que l'on a initialisé l'attribut « nbcomptes ».
- Et là, tout étudiant normalement constitué doit se dire que pour que l'exemple proposé ait un sens, il faut rajouter dans chaque constructeur la commande

```
nbcomptes++;
```

Ainsi, à chaque nouvel objet on incrémentera l'attribut « nbcomptes ».

## 5. Programmation des classes et objets

### ➤ Destructeurs

Le destructeur est **une fonction membre** appelée automatiquement lors de la destruction d'une instance d'une classe( c-à-d un objet ).

- Le destructeur sert à récupérer la place mémoire (d'un objet ) allouée dynamiquement.

➤ *Un seul destructeur par classe*

➤ Il porte le même nom que la classe précédée du symbole ~

➤ Il ne reçoit aucun argument et ne retourne aucune valeur



## 5. Programmation des classes et objets

- Destructeurs: **Quand créer un destructeur ?**
  - Quand la classe contient un attribut qui est un pointeur.
  - Quand on veut que la libération de la mémoire s'accompagne d'une autre tâche (affichage, modification d'un attribut de classe...).

# 5. Programmation des classes et objets

## ➤ Destructeurs: **Propriétés d'un destructeur**

- Le nom d'un destructeur est le même que celui de la classe précédé d'un « ~ » .
- Les destructeurs ne retournent aucune valeur (même pas void).
- Une classe ne peut posséder qu'un seul destructeur.

## ➤ Remarques

- Si le programmeur ne crée pas lui même un destructeur dans une classe, le compilateur en génère un **par défaut**.
- Ce dernier remplira tout à fait son rôle, à l'unique condition que la **classe ne possède pas d'attributs de type pointeurs**, ou que ses méthodes n'en utilisent pas.
- Sinon, il faudra en créer un, afin de libérer l'espace mémoire occupée par ces pointeurs.

## 5. Programmation des classes et objets

### ➤ Destructeurs: **Exemple**

- on veut mettre à jour le nombre de comptes créés à l'aide du destructeur
- Dans le « .h » déclaration du destructeur :

```
~Compte();
```

- Dans le « .cpp » implémentation de ce destructeur :

```
Compte::~~Compte()  
{ cout << "et zou un de moins !" << endl;  
  nbcomptes--; }
```

## 5. Programmation des classes et objets

- Destructeurs: **suite de l'exemple**
- le « main » et son résultat.

```
int main()
{ Compte::affichernbcomptes();
  { Compte c;
    Compte::affichernbcomptes(); }
  Compte::affichernbcomptes();
}
```

```
nombre de comptes : 0
nombre de comptes : 1
et zou un de moins ?
nombre de comptes : 0
```

# 5. Programmation des classes et objets

## ➤ Fonctions amies

- ❑ Une fonction **amie** est une fonction **non membre** d'une classe et possédant le droit d'accès aux membres privés de ses objets.
- ❑ **Intérêt** : résoudre certains problèmes de communication entre objets
- ❑ Fonction qui a besoin, dans certains cas, d'accéder aux membres privés de deux classes différentes
- ❑ **Attention !!** : Violation du concept d'encapsulation
- ❑ L'amitié est déclarée par le mot clé **friend**

## 5. Programmation des classes et objets

### ➤ Fonctions indépendante et amie d'une classe

#### ☐ *Syntaxe :*

```
class A {  
    ...  
    public:  
    ...  
    friend type_retour fonct_amie(...);  
    ...  
};  
  
// définition de la fonction amie  
type_retour fonct_amie(...)  
{  
    ...  
}
```

#### ☐ *Remarque:*

```
// Attention !!!! Fausse définition  
type_retour A::fonct_amie(...)  
{  
    ...  
}
```

## 5. Programmation des classes et objets

- Fonction membre d'une classe A et amie d'une ou plusieurs classes

□ *Syntaxe :*

```
class A {  
    ...  
    public:  
    ...  
    type_retour fonct_amie(...);  
    ...  
};  
class B {  
    ...  
    public:  
    ...  
    friend type_retour A::fonct_amie(...);  
};
```

## 5. Programmation des classes et objets

- Toutes les fonctions membres d'une classe A sont amies d'une autre classe B
- *Syntaxe* :

```
class A {  
    ...  
    public:  
    ...  
};  
class B {  
    ...  
    public:  
    ...  
    friend class A; // les fonctions membres de A sont amies de  
                    // classe B  
};
```



# 5. Programmation des classes et objets

## ➤ Fonctions et attributs static

- Le terme **static** applicable aux méthodes et aux attributs
- Stockage dans la zone statique
- Durée de vie permanente, égale à celle du programme
- Initialisation au moment de la compilation, initialisation par défaut à 0

- Syntaxe

**static float x;**

# 5. Programmation des classes et objets

## ➤ Fonctions et attributs static

❑ Un membre **static** est partagé par **tous les objets de la classe**

❑ *Syntaxe*: Le membre est précédé par le mot clé static lors de la déclaration de la classe

```
static type nom_var;
```

❑ L'initialisation n'est pas automatique

```
Type class_name::ident = 0;
```

## 5. Programmation des classes et objets

### ➤ Fonctions et attributs static

#### ☐ Exemple

```
#include<iostream.h>
class point {
    static int nb;
    int x;
    int y;

    public:
    point(int,int);
    void deplace(int,int);
    void affiche();
};

int point::nb=0;

point::point(int abs,int ord)
{
    nb++;
    x=abs;
    y=ord;
}
```

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

void point::affiche()
{
    cout << "position :" <<x <<"," <<y
    <<"nombre d'objets" << nb <<"\n";
}

void main()
{
    point a(10,20);
    a.affiche();
    a.deplace (5,5);
    a.affiche();
    point b(30,30);
    b.affiche(); }
```

# 5. Programmation des classes et objets

## ➤ Fonctions et attributs static

### ❑ Fonctions membre static

- ❑ Sont indépendantes de tous les objets de la classe
- ❑ Agissent sur des membres static
- ❑ Sont appelées sans référence à un objet
- ❑ référencées directement par le nom de la classe
- ❑ **nom\_classe::fonct\_static(...);**
- ❑ Peuvent être appelées avant la création de tout objet
- ❑ Syntaxe:

```
class nom_classe {  
    ....  
    public:  
    ...  
    static type fonct_static(...);  
};
```

## 5. Programmation des classes et objets

### ➤ Fonctions et attributs static

#### ❑ Exemple

```
#include<iostream.h>
class point {
    static int nb;
    int x;
    int y;

    public:
        point(int,int);
        void deplace(int,int);
        void affiche();
        static void compteur();
};

int point::nb=0;

point::point(int abs,int ord)
{nb++;
x=abs;
y=ord;
}

void point::compteur()
{cout <<"\n nombre de points :"<<nb;
}
```

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

void point::affiche()
{
cout << "position :" <<x <<"," <<y
<<"nombre d'objets" << nb <<"\n";
}

void main()
{
    point a(i,i);
    a.affiche();
    point::compteur();
    for(int i=0;i<4;i++)
        senario(i);

    return 0;
}
```

## 5. Programmation des classes et objets

- Exemple 1

```
class point {  
    int x;  
    int y;  
  
public:  
    point(int,int);  
    void deplace(int,int);  
    friend int complexe::coincide(point&);  
    void affiche();  
};
```

```
class complexe {  
    int x;  
    int y;  
public:  
    complexe(int,int);  
    int coincide(point&);  
    void affiche();  
    ...  
};
```

## 5. Programmation des classes et objets

- Exemple 2:

```
class point {  
    int x;  
    int y;  
  
public:  
    point(int,int);  
    void deplace(int,int);  
    friend int coincide(complexe&,  
point&);  
    void affiche();  
};
```

```
class complexe {  
    int x;  
    int y;  
public:  
    complexe(int,int);  
    int coincide(point&);  
friend int coincide(complexe&,point&);  
    void affiche();  
    ...  
};
```

*Coincide est une fonction indépendante*

# 5. Programmation des classes et objets

## ➤ Surcharge des opérateurs

- **Opérateur?**

- Opération entre des **opérandes**.
- Un appel à un opérateur est similaire à un appel de fonctions.
- A **Op** B se traduit par **A.fonction(B)**
- **Exemple:**
  - **a + b a.somme(b)**
  - **a = b a.affectation(b)**
  - **++a a.incrementation()**



# 5. Programmation des classes et objets

## ➤ Surcharge des opérateurs

- Alternative aux fonctions membre
  - Utilisation plus naturelle que les fonctions
  - Exemple : **somme de deux complexes** c1 et c2
  - $c3 = c1 + c2$  **au lieu** de  $c3=c1.somme(c2)$
- Etendre la sémantique d'un opérateur de type de base aux nouveaux types utilisateurs (instances de classes)

# 5. Programmation des classes et objets

## ➤ Surcharge des opérateurs

- Opérateurs à surcharger
  - Opérateurs arithmétiques  
+ - \* / %
  - Opérateurs d'incrément/décémentation  
++, --
  - Opérateurs de comparaison  
< > >= <= == !=
  - Opérateur d'affectation  
=
  - ...

# 5. Programmation des classes et objets

## ➤ Surcharge des opérateurs

- Syntaxe

- Comme la définition des fonctions, sauf on rajoute devant l'opérateur le mot clé **operator**

```
type_retour classe :: operator op([parm1],...)  
{  
    ...  
}
```

## 5. Programmation des classes et objets

- Surcharge des opérateurs
- Exemple

```
#include<iostream.h>
class complexe {
    int reel;
    int img;
public:
    void saisir(int,int);
    complexe operator+ (complexe &);
    void affiche();
};
void complexe::saisir(int r,int i)
{
    reel=r;
    img=i;
}
void complexe::affiche()
{
    cout << " partie réelle :" << reel
    cout <<" , partie imaginaire :" <<img
    <<"\n";
}
complexe complexe::operator+(complexe & c)
{
    complexe z;
    z.reel=reel+c.reel;
    z.img=img+c.img;
    return z;
}
void main()
{
    complexe z1,z2,z3;
    z1.saisir(1,1);
    z2.saisir(2,2);
    z3=z1+z2;
    z3.affiche();
}
```

## 5. Programmation des classes et objets

### ➤ Surcharge des opérateurs

- Surcharge de l'opérateur d'affectation
  - L'**opérateur d'affectation** est surchargé comme une **fonction membre**.
  - L'opérateur d'affectation **stocke** le résultat dans l'objet appelant et **retourne** une référence sur cet objet.

## 5. Programmation des classes et objets

- Surcharge des opérateurs
- Surcharge de l'opérateur d'affectation: exemple

```
class rationnel {  
    int a,b;  
    public:  
        rationnel & operator=(const rationnel &)  
};
```

```
rationnel& rationnel::operator=(const rationnel & r)  
{  
    a = r.a; b = r.b;  
    return *this;  
}
```

# 5. Programmation des classes et objets

## ➤ Tableau d'objets

- **Deux possibilités pour déclarer et initialiser un tableau d'objets**
- Utiliser le constructeur par défaut de la classe pour chacun des éléments du tableau.
- Utiliser différents constructeurs ou différents jeux de paramètres pour chacun des éléments du tableau.

## 5. Programmation des classes et objets

### ➤ Tableau d'objets

- **Exemple**

- Chaque élément initialisé avec le constructeur par défaut :

```
Compte tab[3];
```

- Chaque élément initialisé différemment :

```
Compte tab[3] = { Compte(100,1,"compte courant"),  
                 Compte(200,2,"livret A"), Compte(500,3,"livret B")};
```



## 5. Programmation des classes et objets

### ➤ Tableau d'objets

- **Accès aux membres** : comme dans le cas d'un seul objet.
- **Exemple** :

```
int main()
{ Compte tab[3] = { Compte(100,1,"compte courant"),
                  Compte(200,2,"livret A"), Compte (500,3,"livret B")};
  tab[0].affichersolde();
  tab[1].affichernumero();
  tab[2].affichernom(); }
```

# 5. Programmation des classes et objets

## ➤ Tableau d'objets

- **Création dynamique d'un objet.**
- En utilisant le constructeur par défaut :
  - *NomDeLaClasse \*pt;*
  - *pt = new NomDeLaClasse;*
- En utilisant un autre constructeur :
  - *NomDeLaClasse \*pt;*
  - *pt = new NomDeLaClasse( paramètres );*

# 5. Programmation des classes et objets

## ➤ Tableau d'objets

- **Accès aux membres :**
- Avec l'opérateur « . » :
  - (\*pt).NomDeLaMéthode( ... )
- Avec l'opérateur « -> » :
  - pt->NomDeLaMéthode( ... )

## 5. Programmation des classes et objets

### ➤ Tableau d'objets

- **Exemple 1 :**

```
int main()
{
    Compte *pt;
    pt = new Compte;
    (*pt).affichersolde();
    pt->affichernumero();
    pt->affichernom(); }
}
```

```
solde : 0
numero de compte : 0
nom du compte : NoName
```

## 5. Programmation des classes et objets

### ➤ Tableau d'objets

- **Exemple 2 :**

```
int main()
{ Compte *pt;
  pt = new Compte(100000,666,"Laurent");
  (*pt).affichersolde();
  pt->affichernumero();
  pt->affichernom(); }
```

```
solde : 100000
numero de compte : 666
nom du compte : Laurent
```

# 5. Programmation des classes et objets

## ➤ Tableau d'objets

- **Destruction d'un objet dynamique:**
  - Quand un objet créé dynamiquement n'est plus d'utilité, on le supprime de la mémoire avec l'opérateur « **delete** ».
  - L'opérateur « **delete** » appellera alors automatiquement le destructeur de la classe.
  - Avec les notations précédentes, cela donne :  
**delete** pt;

# 5. Programmation des classes et objets

## ➤ Tableau d'objets

- **Création dynamique d'un tableau d'objets:**

- On est obligé de recourir au constructeur par défaut de la classe :

```
NomDeLaClasse *tab;
```

```
tab = new NomDeLaClasse[Taille];
```

- En effet, toute utilisation d'une liste d'initialisation est impossible.

## 5. Programmation des classes et objets

### ➤ Tableau d'objets

- **Exemple:**

```
int main()
{ Compte *tab;
  tab = new Compte[3];
  tab[0].affichersolde();
  tab[1].affichernumero();
  tab[2].affichernom(); }
```

```
solde : 0
numero de compte : 0
nom du compte : NoName
```



# 5. Programmation des classes et objets

## ➤ Tableau d'objets

- **Destruction du tableau dynamique**
- À la fin de son usage, on détruira notre tableau dynamique à l'aide de l'opérateur « **delete [ ]** ».
- Avec les notations précédentes, cela donne :

**delete [ ] pt;**