



P00 – Langage C++

Les fonctions amies

Surcharge/ Surdéfinition des opérateurs

(PARTIE 1)

1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

Introduction

- Une fonction amie d'une classe est une fonction extérieure de la classe (non membre de la classe), et qui a accès aux membres **privés** de la classe.
- L'amitié est déclarée en utilisant le mot-clé réservé: **friend**.
- une telle déclaration d'amitié **autorise** une fonction extérieure à **accéder aux membres privés** de la classe, au même titre que n'importe quelle fonction membre.
- L'emplacement de la déclaration d'amitié au sein de la classe est absolument indifférent.

plan

- **Exemple de fonction indépendante amie d'une classe**
- **Les différentes situations d'amitié**
 - Fonction indépendante amie d'une classe
 - Fonction membre d'une classe, amie d'une autre classe
 - Fonction amie de plusieurs classes
 - Toutes les fonctions d'une classe sont amies d'une autre classe

```

void coincider(point a, point b)
{
    if(a.x==b.x && a.y==b.y)
        cout<<"coincident "<<endl;
    else cout<<"ne coincident pas "<<endl;
}

void main()
{
    point a;
    point b(12,12);
    coincider(a,b);
    system("PAUSE");
}

```

```

(Global Scope)
class point
{
protected:
    int x;
    int y;
public:

    point(int =99,int =99);
    virtual void afficher(string = "");
    virtual ~point();
    virtual void saisir_point();
    bool coincide(point);
    friend void coincider(point,point);
};

```

Quand est ce qu'une fonction doit être amie d'une classe

- Une fonction amie d'une classe:
 - possède un ou plusieurs arguments du type de la classe
 - utilise une variable locale de cette classe (qui peut être retournée par la fonction).

```
void fct ( point a, .....)  
{  
    point b; // variable locale non  
    retournée par la fonction  
    .....  
}
```

```
point fct ( point a, .....)  
{  
    point b; // variable locale  
    retournée par la fonction  
    .....  
    return b;  
}
```

plan

- **Exemple de fonction indépendante amie d'une classe**
- **Les différentes situations d'amitié**
 - Fonction indépendante amie d'une classe
 - Fonction membre d'une classe, amie d'une autre classe
 - Fonction amie de plusieurs classes
 - Toutes les fonctions d'une classe sont amies d'une autre classe

Fonction indépendante amie d'une classe

// fichier mes_fcts.h

void fct (A....); // classe A

// fichier mes_fcts.cpp

void fct (A)

{

...

Accès aux membres privés de
tout objet de type A.

}

class A

{

....

friend void fct (A....);

}

plan

- **Exemple de fonction indépendante amie d'une classe**
- **Les différentes situations d'amitié**
 - Fonction indépendante amie d'une classe
 - **Fonction membre d'une classe, amie d'une autre classe**
 - Fonction amie de plusieurs classes
 - Toutes les fonctions d'une classe sont amies d'une autre classe

Fonction membre d'une classe, amie d'une autre classe

```
class A
{
    ....
    friend int B:: fct (A);
}
```

```
class A;
class B
{
    ....
    int fct (A);
    // fct accède aux membres privés de
    // tout objet de type A
}
```

- Il faut préciser, dans la déclaration d'amitié, la classe à laquelle appartient la fonction concernée, à l'aide de l'opérateur de résolution de portée (::).

plan

- **Exemple de fonction indépendante amie d'une classe**
- **Les différentes situations d'amitié**
 - Fonction indépendante amie d'une classe
 - Fonction membre d'une classe, amie d'une autre classe
 - **Fonction amie de plusieurs classes**
 - Toutes les fonctions d'une classe sont amies d'une autre classe

Fonction amie de plusieurs classes

- Une fonction (indépendante ou membre) peut être amie dans différentes classes.

```
class B;  
class A  
{  
    ....  
    friend void fct (A, B);  
}
```

```
class A;  
class B  
{  
    ....  
    friend void fct (A, B);  
}
```

```
void fct (A....., B.....) // ici, fct est indépendante  
{ // accès aux membres privés de tout objet de type A ou B  
}
```

plan

- **Exemple de fonction indépendante amie d'une classe**
- **Les différentes situations d'amitié**
 - Fonction indépendante amie d'une classe
 - Fonction membre d'une classe, amie d'une autre classe
 - Fonction amie de plusieurs classes
 - **Toutes les fonctions d'une classe sont amies d'une autre classe**

Classe amie

- Il est possible d'effectuer autant de déclarations d'amitié qu'il y a de fonctions concernées.
- Mais il est plus simple d'effectuer une déclaration globale.
- Ainsi, pour dire que **toutes les fonctions membres de la classe B sont amies de la classe A**, on placera, dans la classe A, la déclaration :

```
class A
{
    friend class B;
    ....
}
```

```
class B
{
    ....
}
```

L'amitié n'est ni symétrique ni transitive

- (la classe A est amie de la classe B) **n'implique pas** (la classe B est amie de la classe A) → pas de symétrie.
- (la classe A est amie de la classe B et la classe B est amie de la classe C) **n'implique pas** (la classe A est amie de la classe C) → pas de transitivité.
- D'une manière générale:
Utiliser une **fonction membre** quand vous pouvez, et une fonction amie quand vous êtes dans l'obligation de le faire.

La Surdéfinition/ Surcharge des opérateurs

Les opérateurs en C++

- Les opérateurs en C++:
 - Opérateurs arithmétiques (+ - * / % etc),
 - Opérateurs << et >> ,
 - etc
- Ces opérateurs sont **surdéfinis** pour les types de base: char, short, int, long, float, double, char*.
- En langage C++, il est possible de **surdéfinir** ces opérateurs pour les classes.
- Dans une classe, Les opérateurs peuvent être surdéfinis comme:
 - **fonction membres** (+, [], etc)
 - ou **fonctions extérieures (amies)** (operator<<, operator>>, etc).

- **Qui peut être surdéfini?**

On peut redéfinir une quarantaine d'opérateurs:

- `+ - * / % ^ & |`
- `->` opérateur de sélection de membre via pointeur
- `[]` opérateur d'indexation
- `()` opérateur d'appel de fonction
- `new` opérateur d'allocation de mémoire dynamique
- `delete` opérateur de désaffectation de mémoire dynamique

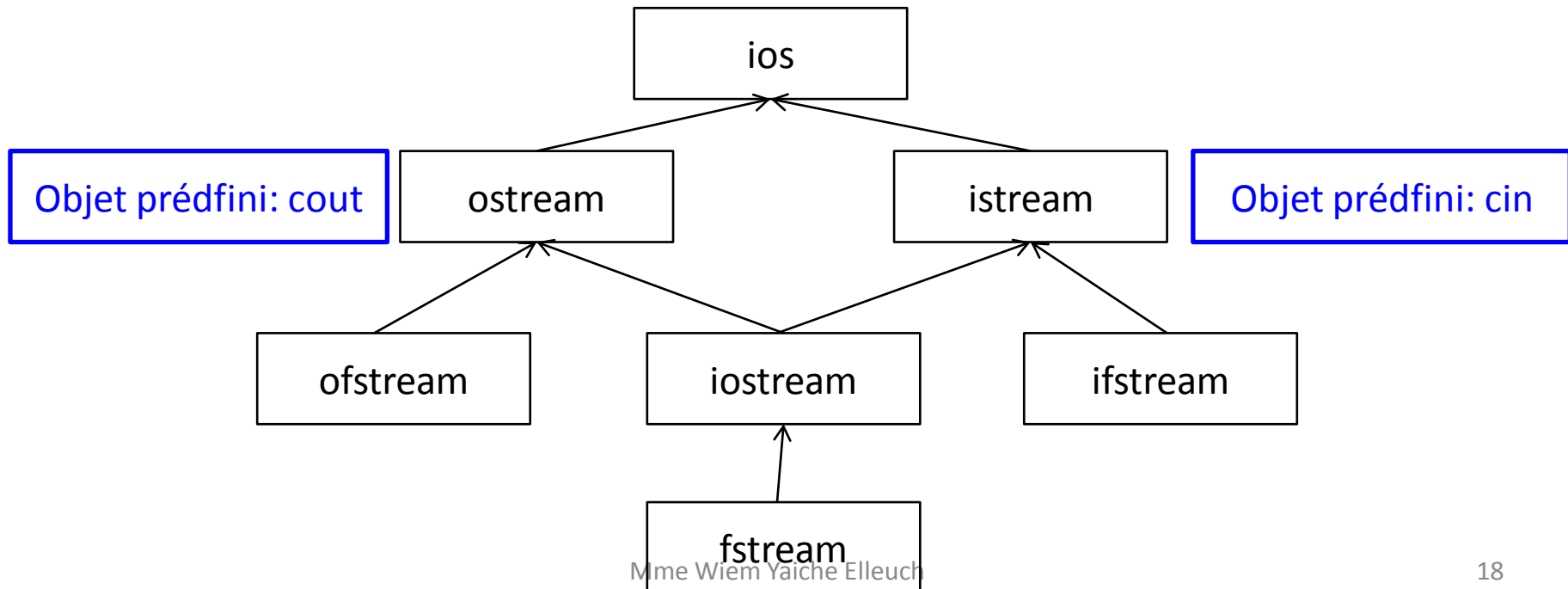
- **Qui ne peut pas être redéfini?**

les opérateurs suivants ne peuvent pas être surchargés (une liste non exhaustive):

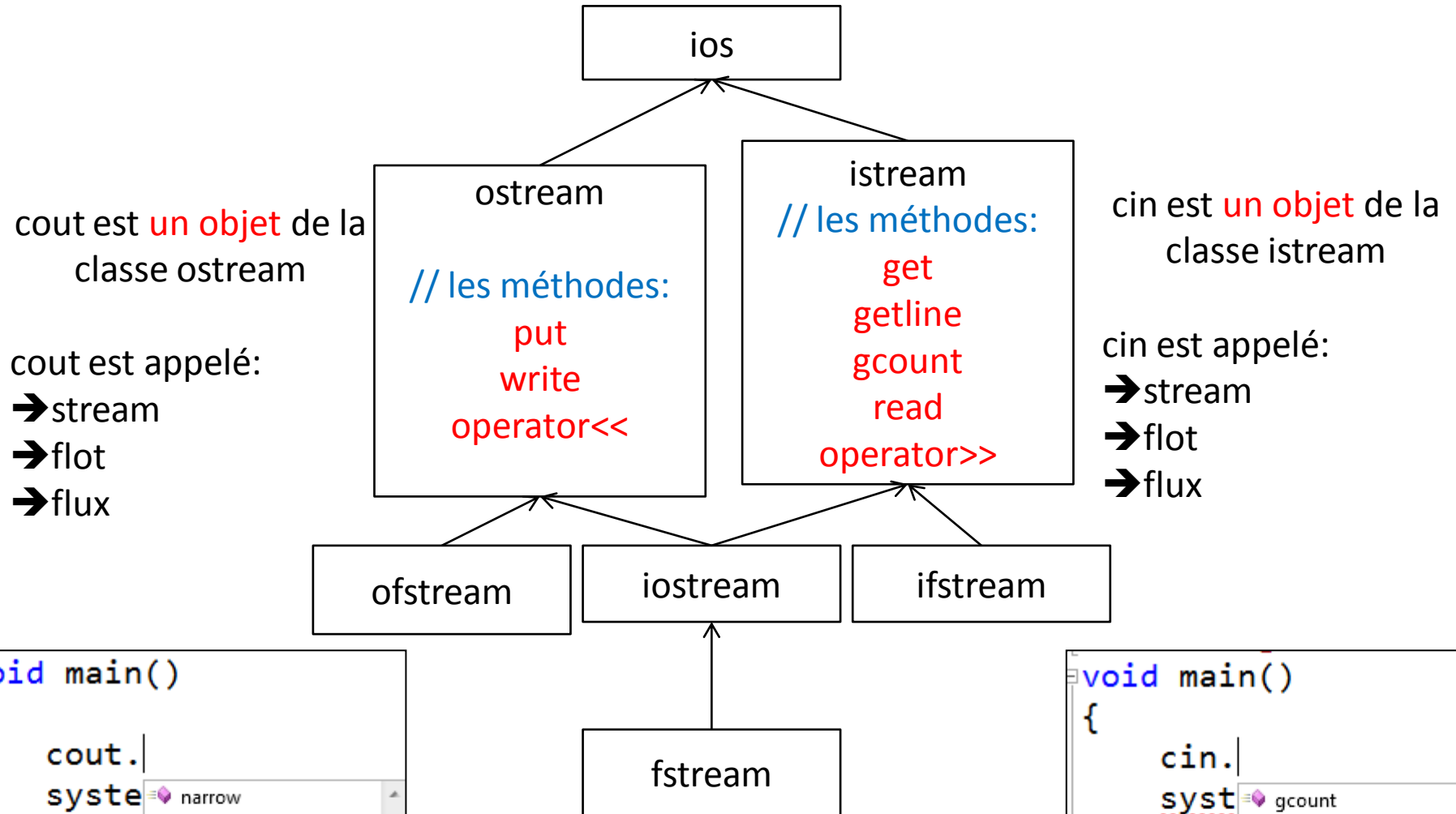
- `.` opérateur de sélection de membre via objet
- `*` opérateur pointeur vers un membre via objet
- `::` opérateur de résolution de portée
- `sizeof` opérateur déterminant la taille en octets

Les entrées/sorties

- **ios** : classe de base des entrées/sorties.
- **istream** : classe dérivée de *ios* pour les flots en entrée.
 - **cin** est un objet prédéfini instance de la classe *istream*
- **ostream** : classe dérivée de *ios* pour les flots en sortie.
 - **cout** est un objet prédéfini instance de la classe *ostream*



Les entrées sorties



```
void main()
{
    cout.
    syste
```

Autocomplete suggestions for `cout.`:

- narrow
- oct
- operator void *
- operator<<
- operator=
- operator!
- opfx
- osfx
- out

```
void main()
{
    cin.
    syst
```

Autocomplete suggestions for `cin.`:

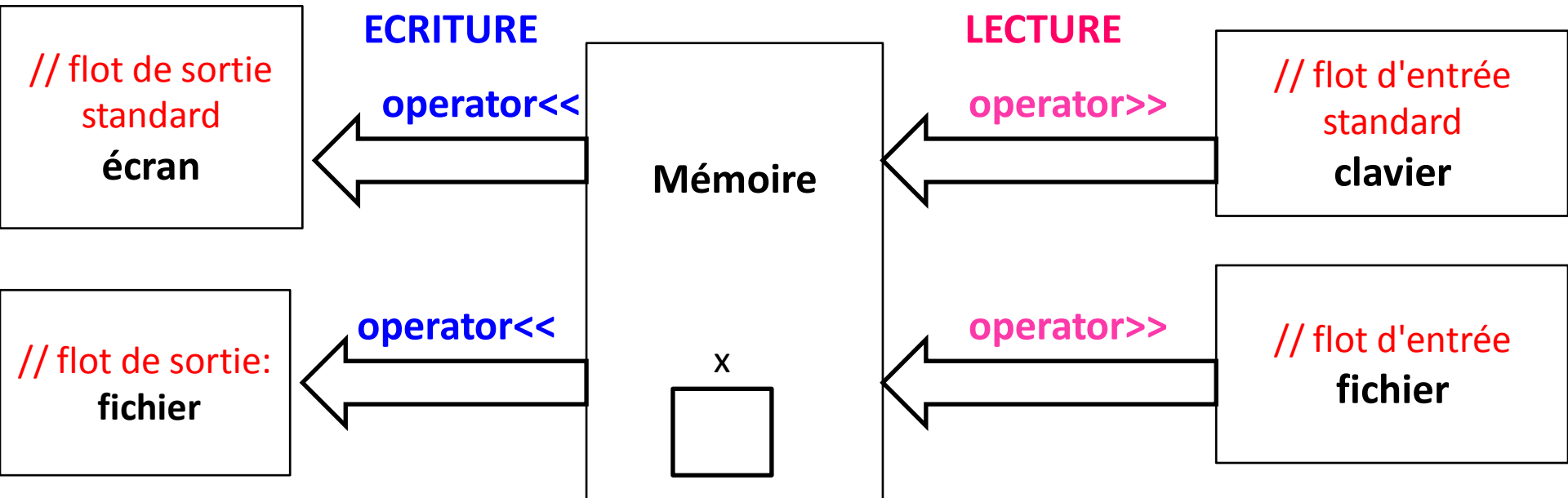
- gcount
- get
- getline
- getloc
- good
- goodbit
- hex
- hexfloat
- ignore

Comment ?

- Les opérateurs ont tous le même préfixe : « **operator** ».
 - Méthode **operator+** pour l'addition
 - Méthode **operator-** pour la soustraction
 - Méthode **operator<<** pour l'affichage
 - Méthode **operator>>** pour la lecture
- ➔ surcharger l'opérateur « + » pour une classe consiste à définir la méthode **operator+**
- Les opérateurs peuvent être définis comme **membres** ou **indépendantes** d'une classe.

```
void main()  
{  
    int x;  
    cin.operator>>(x); // <==> cin>>x;  
    cout.operator<<(x); // <==> cout<<x;  
    system("PAUSE");  
}
```

E: entree / S: sortie



Écriture sur le flot de sortie (écran, fichier, etc): **operator<<** (envoyer dans le flux)²¹
Lecture à partir du flot d'entrée (clavier, fichier, etc): **operator>>** (extraction du flux)

Surcharge des opérateurs

- Les surcharges doivent renvoyer une référence sur une variable de type istream ou ostream pour permettre un enchainement d'opérations de lecture ou écriture.
- Il faut donc renvoyer une référence sur le premier paramètre afin que ce dernier devienne le paramètre de l'opération suivante.

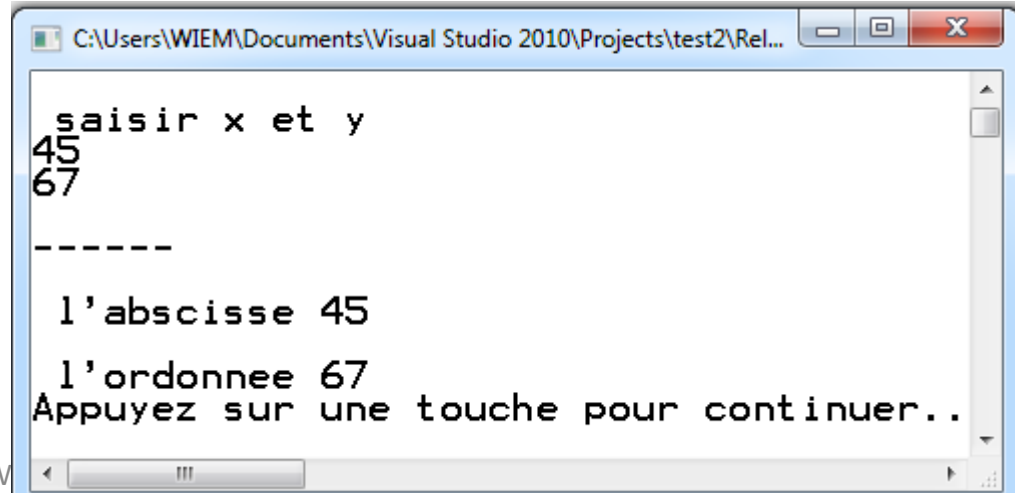
```
istream& operator>> (istream& in, classeX& objet)
{
    /* Lecture de l'objet */
    return in;
}
```

```
ostream& operator<< (ostream& out, const classeX& objet)
{
    /* Ecriture de l'objet */
    return out;
}
```

Surcharge des opérateurs de la classe point

```
class point
{
    int x;
    int y;
public:
    point(int =99,int =99);
    void afficher(string = "");
    ~point();
    void saisir_point();
    bool coincide(point);
    friend ostream& operator<<(ostream&, point&);
    friend istream& operator>>(istream&, point&);
};
```

```
void main()
{
    point a;
    cin>>a;
    cout<<"\n-----"<<endl;
    cout<<a;
    system("PAUSE");
}
```



```
saisir x et y
45
67
-----
l'abscisse 45
l'ordonnee 67
Appuyez sur une touche pour continuer..
```

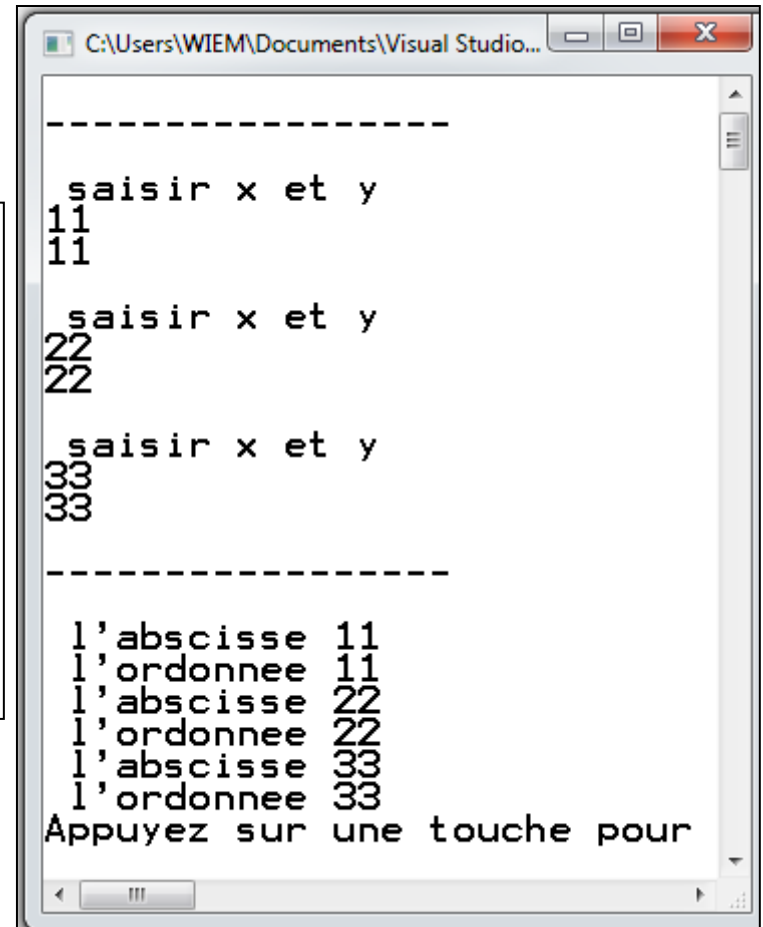
Surcharge des opérateurs de la classe point

```
ostream& operator<< (ostream& out, point& pt)
{
    // out<<"\n debut surcharge operator<< point "<<endl;
    out<<"\n l'abscisse "<<pt.x;
    out<<"\n l'ordonnee "<<pt.y;
    // out<<"\n FIN surcharge operator<< point "<<endl;
    return out;
}
```

```
istream& operator>> (istream& in, point &pt)
{
    // cout<<"\n debut surcharge operator>> point "<<endl;
    cout<<"\n saisir x et y "<<endl;
    in>>pt.x;
    in>>pt.y;
    // cout<<"\n FIN surcharge operator>> point "<<endl;
    return in;
}
```


Enchainements des lectures et des écritures

```
void main()
{
    point a,b,c;
    cout<<"\n-----"<<endl;
    cin>>a>>b>>c;
    cout<<"\n-----"<<endl;
    cout<<a<<b<<c;
    cout<<endl;
    system("PAUSE");
}
```



```
-----
saisir x et y
11
11
saisir x et y
22
22
saisir x et y
33
33
-----
l'abscisse 11
l'ordonnee 11
l'abscisse 22
l'ordonnee 22
l'abscisse 33
l'ordonnee 33
Appuyez sur une touche pour
```

Surcharge des opérateurs de lecture et d'écriture de la classe etudiant

```
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);
    friend ostream& operator<< (ostream&, etudiant&);
    friend istream& operator>> (istream&, etudiant&);
};

void main()
{
    etudiant a;
    cin>>a;
    cout<<"\n";
    cout<<a;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test2\Release\te...

```
saisir code, nom et nbre de notes
999
ali
2

saisir les notes
16
17

-----

le code est 999
le nom est ali
le nbre de notes 2

les notes sont
16 17
la moyenne est 16.5
Appuyez sur une touche pour continuer...
```

Surcharge des opérateurs de la classe etudiant

```
ostream& operator<< (ostream& out, etudiant& e)
{
    cout<<"\n surcharge operator<< "<<endl;
    out<<"\n code: "<<e.code;
    out<<"\n nom: "<<e.nom;
    out<<"\n nbre notes "<<e.nb_notes;
    out<<"\n les notes sont "<<endl;
    for(int i=0; i<e.nb_notes; i++)
        out<<e.notes[i]<<" ";
    out<<"\n moyenne: "<<e.moyenne<<endl;
    cout<<"\n FIN surcharge operator<< "<<endl;
    return out;
}
```

```
istream& operator>> (istream& in, etudiant& e)
{
    cout<<"\n surcharge operator>> "<<endl;
    in>>e.code;
    in>>e.nom;
    in>>e.nb_notes;
    e.notes=new float[e.nb_notes];
    cout<<"\n saisir les notes "<<endl;
    for(int i=0; i<e.nb_notes; i++)
        in>>e.notes[i];
    e.calcul_moyenne();
    cout<<"\n FIN surcharge operator>> "<<endl;
    return in;
}
```

Classe etudiant avec un tableau dynamique (notes) et une matrice dynamique (mat)

```
class etudiant
{
    int ce;
    string nom;
    int nbNotes;
    float *notes;
    float moyenne;
    int l;
    int c;
    int **mat;

public:
    friend ostream& operator<< (ostream&, etudiant&);
    friend istream& operator>> (istream&, etudiant&);

    etudiant(int =99, string = "", int=1);
    etudiant(const etudiant&);
    void afficher(string = "");
    void saisirNotes();
    void calculMoyenne();
};
```

```
istream& operator>> (istream& in, etudiant& etd)
{
    //cout<<"\n saisir ce, nom et nbNotes "<<endl;
    in>>etd.ce;
    in>>etd.nom;
    in>>etd.nbNotes;
    etd.notes=new float[etd.nbNotes];
    for(int i=0; i<etd.nbNotes;i++)
        in>>etd.notes[i];
    // cout<<"\n saisir moyenne, l et c "<<endl;
    in>>etd.moyenne;
    in>>etd.l;
    in>>etd.c;
    etd.mat=new int* [etd.l];
    for(int i=0; i<etd.l; i++)
        etd.mat[i]=new int [etd.c];

    for(int i=0; i<etd.l;i++)
        for(int j=0; j<etd.c; j++)
            in>>etd.mat[i][j];
    return in;
}
```

```
ostream& operator<< (ostream& out, etudiant &etd)
{
    //cout<<"\n surcharge << "<<endl;
    out<<"\n ce: "<<etd.ce<<endl;
    out<<"\n nom: "<<etd.nom<<endl;
    out<<"\n nbnotes "<<etd.nbNotes<<endl;
    for(int i=0; i<etd.nbNotes; i++)
        out<<etd.notes[i]<<" ";
    out<<endl;
    out<<"\n la moyenne "<<etd.moyenne<<endl;
    out<<"\n l et c : "<<etd.l<<" "<<etd.c<<endl;

    for(int i=0; i<etd.l; i++)
    {
        for(int j=0; j<etd.c; j++)
            out<<etd.mat[i][j]<<" ";
        out<<endl;
    }
    return out;
}
```

Cas de l'héritage

Surcharge des opérateurs << et >> dans la classe dérivée pointCouleur

```
class pointCouleur:public point
{
protected:
    int couleur;
public:
    pointCouleur(int =2,int =3,int =4);
    void afficher(string = "");
    ~pointCouleur(void);
    void saisir_point();
    friend ostream& operator<< (ostream&, pointCouleur&);
    friend istream& operator>> (istream&, pointCouleur&);
};
```

```
ostream& operator<< (ostream& out, pointCouleur& pt)
{
    point *q=&pt;
    out<<*q; // appel de operator<< de la classe point
            // affichage de x et y
    out<<"\n couleur "<<pt.couleur<<endl;
    return out;
}
```

```
istream& operator>> (istream &in, pointCouleur &pt)
{
    point *q=&pt;
    in>>*q; // appel de operator>> de la classe point;
            // saisie de x et y
    in>>pt.couleur;
    return in;
}
```

Cas de l'héritage

Surcharge des opérateurs << et >> dans la classe dérivée

pointColoreMasse

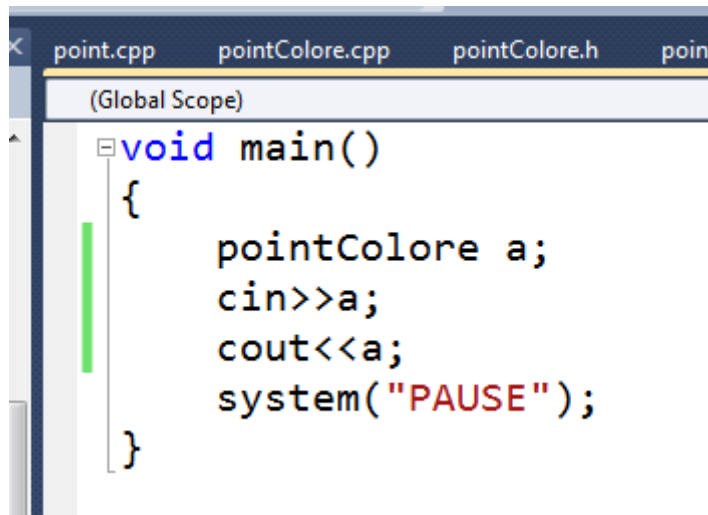
```
(Global Scope)
#pragma once
#include "pointColore.h"
class pointColoreMasse:public pointColore
{
    int masse;
public:
    pointColoreMasse(int =2,int =3,int =4,int =5);
    void afficher(string = "");
    ~pointColoreMasse(void);
    void saisir_point();
    friend ostream& operator<< (ostream&, pointColoreMasse&);
    friend istream& operator>> (istream&, pointColoreMasse&);
};

ostream& operator<< (ostream& out, pointColoreMasse& pt)
{
    pointColore *q=&pt;
    out<<*q; // appel de operator<< de la classe pointColore
            // affichage de x, y e couleur
    out<<"\n masse "<<pt.masse<<endl;
    return out;
}

istream& operator>> (istream &in, pointColoreMasse &pt)
{
    pointColore *q=&pt;
    in>>*q; // appel de operator>> de la classe pointColore;
            // saisie de x, y et couleur
    in>>pt.masse;
    return in;
}
```

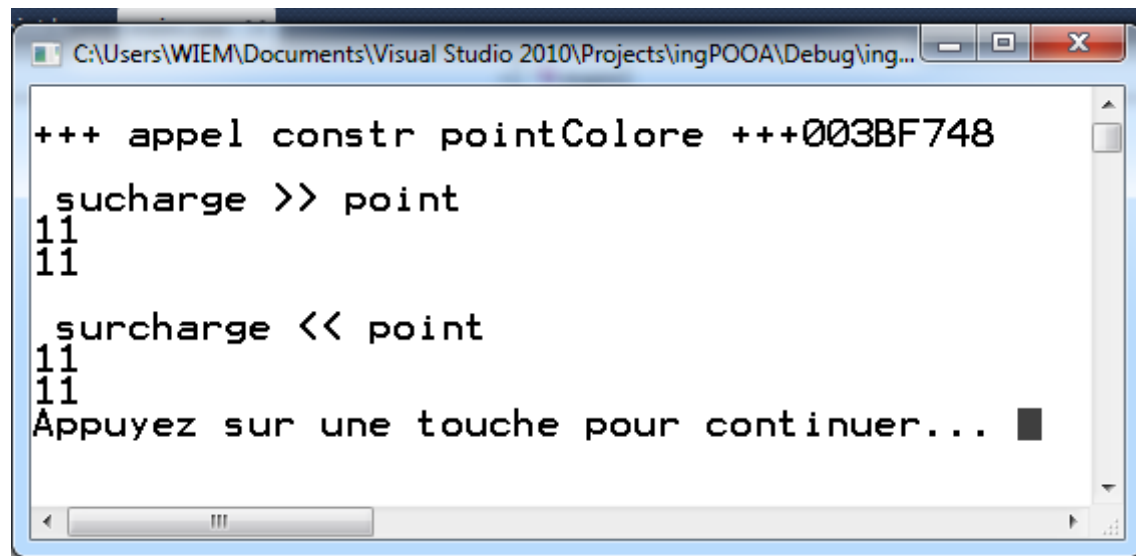

Remarque1

- Si `operator>>` et `operator<<` sont surchargés dans la classe `point` uniquement (ils ne sont pas surchargés dans la classe `pointCouleur`) → un objet `pointCouleur` appellera `operator>>` et `operator<<` de la classe `point` (lecture et affichage de `x` et `y` seulement sans la couleur).
- **Un objet d'une classe dérivée peut utiliser les fonctions amies de la classe mère**



```
point.cpp  pointCouleur.cpp  pointCouleur.h  point...
(Global Scope)

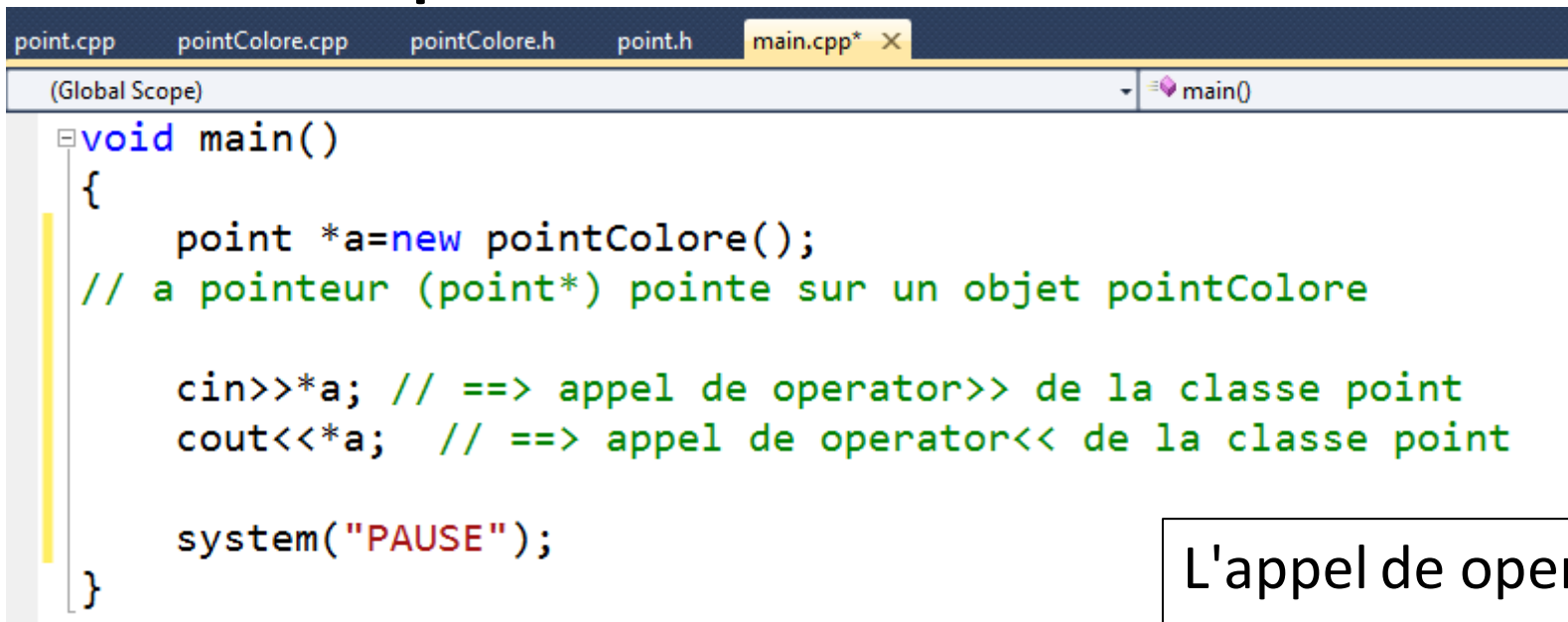
void main()
{
    pointCouleur a;
    cin>>a;
    cout<<a;
    system("PAUSE");
}
```



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

+++ appel constr pointCouleur +++003BF748
surchage >> point
11
11
surchage << point
11
11
Appuyez sur une touche pour continuer...
```

Remarque 2



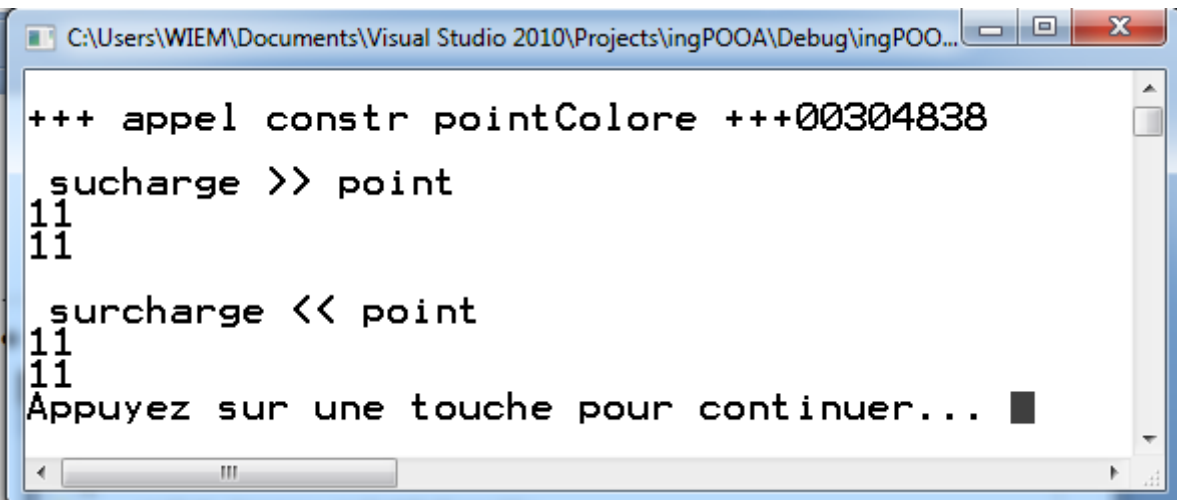
```
point.cpp  pointColore.cpp  pointColore.h  point.h  main.cpp* X
(Global Scope)  main()

void main()
{
    point *a=new pointColore();
    // a pointeur (point*) pointe sur un objet pointColore

    cin>>*a; // ==> appel de operator>> de la classe point
    cout<<*a; // ==> appel de operator<< de la classe point

    system("PAUSE");
}
```

L'appel de operator<< et operator>> depend du type du pointeur et non du type de l'objet pointé



```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\ingPOOA\Debug\ingPOO...
+++ appel constr pointColore +++00304838
surchage >> point
11
11
surchage << point
11
11
Appuyez sur une touche pour continuer... █
```

Question: Dans ce cas, est ce qu'on peut déclarer operator>> et operator<< **virtual** dans la classe point?

Réponse: NON

```
#include<string>
class point
{
protected:
    int x;
    int y;
public:
    friend virtual ostream& operator<< (ostream&, point&);
    friend virtual istream& operator>> (istream&, point&);
}
```

Output

Show output from: Build

\documents\visual studio 2010\projects\ingpooa\ingpooa\point.h(11): error C2575: 'operator <<' : only member functions and bases can be virtual
\documents\visual studio 2010\projects\ingpooa\ingpooa\point.h(12): error C2575: 'operator >>' : only member functions and bases can be virtual

- Les fonctions amies ne peuvent pas être virtuelles → uniquement les fonctions membres d'une classe peuvent être virtuelles