



P00 – Langage C++
Les fonctions amies
Surcharge/Surdéfinition des
opérateurs (PARTIE 2/2)
1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

Surcharge de l'opérateur d'addition (operator+)

- Surcharge en tant que fonction membre
 - `point operator+(point&);` // cas1
 - `int operator+(point&);` // cas2
 - `void operator+(point&);` // cas3
- Surcharge en tant que fonction indépendante

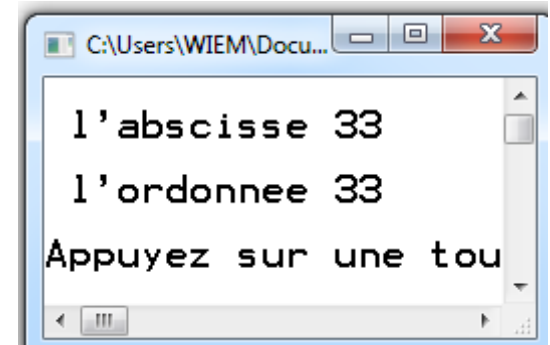
Surcharge operator+en tant que fonction membre: cas1

```
point point::operator+(point& pt)
{
    point res;
    res.x=x+pt.x;
    res.y=y+pt.y;
    return res;
}
```

```
void main()
{
    point a(11,11);
    point b(22,22);
    point c=a+b;
    // point c=a.operator+(b);
    cout<<c<<endl;

    system("PAUSE");
}
```

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



Surcharge operator+ en tant que fonction membre:
cas1 (2^{ème} version): la fonction retourne l'objet ayant le
x le plus grand

```
point point::operator+ (point&pt)
{
    if(x>pt.x) return *this;
    return pt;
}
```

```
void main()
{
    point a(11,11);
    point b(22,22);
    point c=a+b;
    // point c=a.operator+(b);
    cout<<c<<endl;

    system("PAUSE");
}
```

```
class point
{
protected:
    int x;
    int y;
public:
    point(int =99,int =99);
    virtual void afficher(string = "");
    virtual ~point();
    virtual void saisir_point();

    point operator+ (point&);

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

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects...
+++ appel constr point +++ 001FF98C
+++ appel constr point +++ 001FF978
-----
surcharge operator<< point
22
22
Appuyez sur une touche pour continuer.
```

Autre exemple de *this déplacer un point et l'afficher

```
void point::deplacer(int dx, int dy)
{
    x+=dx;
    y+=dy;
    cout<<*this;
    //this: adresse de l'objet qui appelle deplacer
    //*this: l'objet qui appelle deplacer
}
```

(Global Scope)

```
void main()
{
    point a(11,11);

    a.deplacer(100,100);

    system("PAUSE");
}
```

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

```
+++ appel constr point +++ 0027FB08
surcharge operator<< point
111
111
Appuyez sur une touche pour continuer...
```

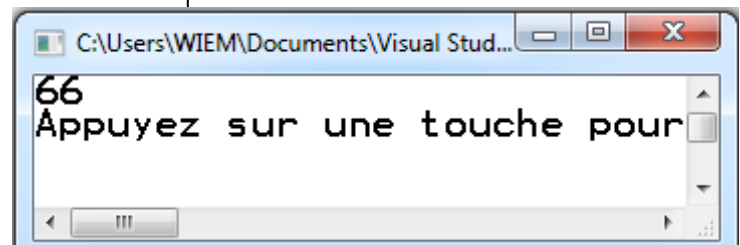
Surcharge operator+ en tant que fonction membre: cas2

```
int point::operator+(point& pt)
{
    int s=0;
    s=x+y+pt.x+pt.y;
    return s;
}
```

```
void main()
{
    point a(11,11);
    point b(22,22);
    int som=a+b;
    // int som=a.operator+(b);
    cout<<som<<endl;

    system("PAUSE");
}
```

```
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);
    int operator+(point&);
    friend ostream& operator<<(ostream&, point&);
    friend istream& operator>>(istream&, point&);
};
```



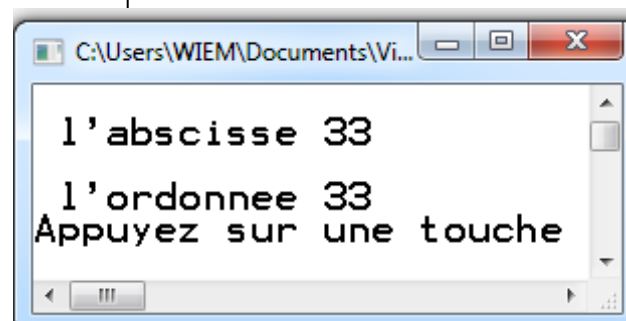
Surcharge operator+ en tant que fonction membre: cas3

```
void point::operator+(point& pt)
{
    x+=pt.x;
    y+=pt.y;
}
```

```
void main()
{
    point a(11,11);
    point b(22,22);
    a+b;
    // a.operator+(b);
    cout<<a;

    system("PAUSE");
}
```

```
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);
    void operator+(point& pt);
    friend ostream& operator<<(ostream&, point&);
    friend istream& operator>>(istream&, point&);
};
```



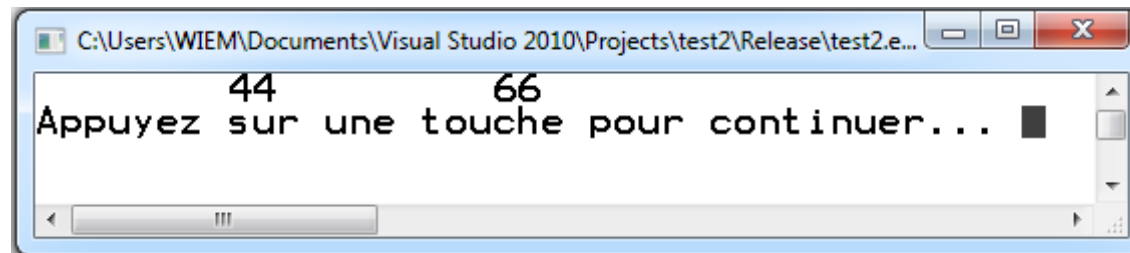
```
C:\Users\WIEM\Documents\Vi...
1'abscisse 33
1'ordonnee 33
Appuyez sur une touche
```

Surcharge operator+ en tant que fonction indépendante

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

```
point operator+(point& a, point& b)
{
    point res;
    res.x=a.x+b.x;
    res.y=a.y+b.y;
    return res;
}

void main()
{
    point a(11,22);
    point b(33,44);
    point c=a+b;
    cout<<c<<endl;
    system("PAUSE");
}
```

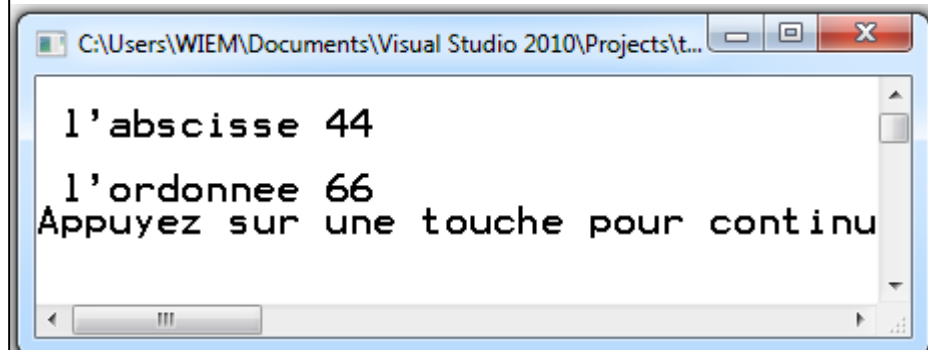


Exemples

```
point point::operator+ (const point& pt)
{
    point res;
    res.x=x+pt.x;
    res.y=y+pt.y;
    return res;
}
```

```
void main()
{
    point a(11,22);
    point b(33,44);
    point c=a+b;
    //point c=a.operator+(b);
    cout<<c;

    system("PAUSE");
}
```

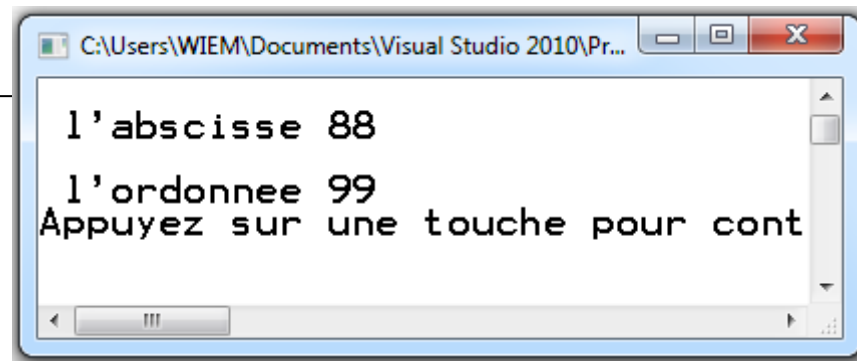


l'abscisse 44
l'ordonnee 66
Appuyez sur une touche pour continu

```
void main()
{
    point a(11,22);
    point b(33,44);
    point c(44,33);

    point d=a+b+c;
    // point d=(a.operator+(b.operator+(c)));
    cout<<d;

    system("PAUSE");
}
```



l'abscisse 88
l'ordonnee 99
Appuyez sur une touche pour cont

Surcharge des opérateurs +, -, ==, <=

```
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);
    point operator+(const point&);
    point operator-(const point&);
    bool operator==(const point&);
    bool operator<=(const point&);
    friend ostream& operator<<(ostream&, point&);
    friend istream& operator>>(istream&, point&);
};
```

Surcharge des opérateurs == et <=

```
bool point::operator<=(const point&pt)
{
    if ((x<=pt.x) && (y<=pt.y)) return true;
    else return false;
}
```

```
bool point::operator==(const point& pt)
{
    if ((x==pt.x) && (y==pt.y)) return true;
    else return false;
}
```

```
void main()
{
    point a(9,8);
    point b(99,88);

    if(a<=b) cout<<"\n inferieur "<<endl;
    else cout<<"\n superieur "<<endl;

    system("PAUSE");
}
```

```
void main()
{
    point a(99,88);
    point b(99,88);

    if(a==b) cout<<"\n egaux "<<endl;
    else cout<<"\n differents "<<endl;

    system("PAUSE");
}
```

Surcharge de l'opérateur -

```
point point::operator-(const point& pt)
{
    point res;
    res.x=x-pt.x;
    res.y=y-pt.y;
    return res;
}
```

```
void main()
{
    point a(99,88);
    point b(33,44);

    point c=a-b;
    // point d=a.operator-(b);
    cout<<c;

    system("PAUSE");
}
```

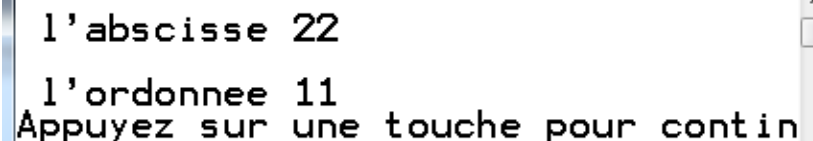


l'abscisse 66
l'ordonnee 44
Appuyez sur une touche pour continue

```
void main()
{
    point a(99,88);
    point b(33,44);
    point c(44,33);

    point d=a-b-c;
    // point d=(a.operator-(b.operator-(c)));
    cout<<d;

    system("PAUSE");
}
```



l'abscisse 22
l'ordonnee 11
Appuyez sur une touche pour contin

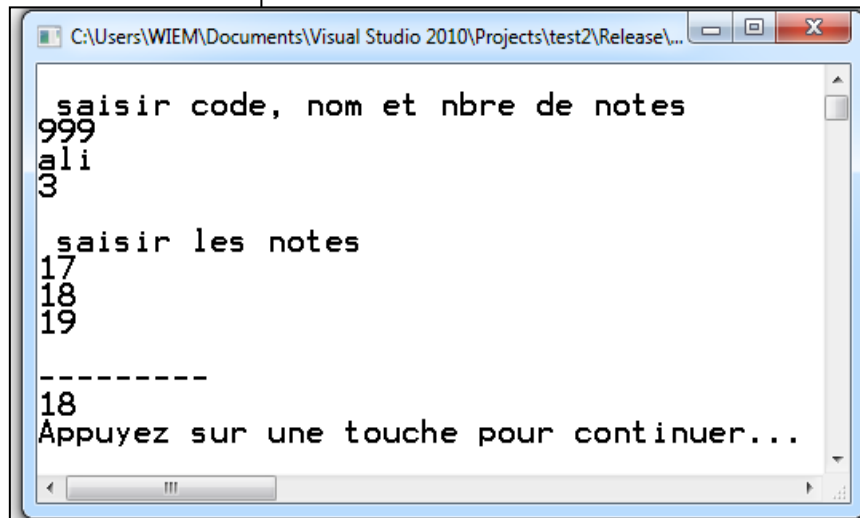
Surcharge de operator[]

```
float etudiant::operator[](int i)
{
    return notes[i];
}
```

```
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);
    float operator[](int);
    friend ostream& operator<< (ostream&, etudiant&);
    friend istream& operator>> (istream&, etudiant&);
};
```

cout<<a.operator[](1)<<endl;

```
void main()
{
    etudiant a;
    cin>>a;
    cout<<"\n-----"<<endl;
    // affichage de la note d'indice 1
    cout<<a[1]<<endl;
    system("PAUSE");
}
```



```
saisir code, nom et nbre de notes
999
ali
3

saisir les notes
17
18
19

-----
18
Appuyez sur une touche pour continuer...
```

Remarques

- Il est possible de surdéfinir l'opérateur `[]` de manière que `a[i]` désigne l'élément d'emplacement `i` de `a`.
- C++ impose de surdéfinir cet opérateur sous la forme d'une **fonction membre**
- C++ interdit de définir l'opérateur `[]` sous la forme d'une fonction amie;
- il en allait déjà de même pour l'opérateur `=`

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

PROBLEME: copie superficielle

SOLUTION: copie profonde

Cas	problème	solution
Affectation de deux objets	Objets dépendants (partagent la même partie dynamique)	Surdéfinition de l'opérateur d'affectation =
Initialisation d'un objet lors de sa déclaration	Objets dépendants (partagent la même partie dynamique)	Constructeur de recopie
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
Objet retourné par une fonction	Libération de la partie dynamique de l'objet à la sortie de la fonction	Constructeur de recopie

surcharge de l'opérateur d'affectation: operator=

Dans main:

```
b=a; // a et b deux objets
```

```
nomClasse & nomClasse::operator= (const nomClasse& w)
{
    if(this !=&w) // éviter le problème de l'instruction (b=b;)
    {
        // copier les attributs "simples" de a dans b
        // libération des tableaux dynamiques liés à l'objet b
        // allocation nouveaux tableaux dynamiques pour b
        // de même taille que ceux de a
        //copie des valeurs des tableaux dynamiques de a dans b

    }
    return *this;
}
```


Sucharge de l'opérateur d'affectation (operator=) 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);
    float operator[](int);
    friend ostream& operator<< (ostream&, etudiant&);
    friend istream& operator>> (istream&, etudiant&);
    etudiant& operator= (const etudiant&);
};
```

Sucharge de l'opérateur d'affectation (operator=) de la classe etudiant

```
// b=a;
etudiant& etudiant::operator= (const etudiant& e)
{
    if (this!=&e)
    {
        code=e.code;
        nom=e.nom;
        nb_notes=e.nb_notes;
        moyenne=e.moyenne;
        // libération ancien espace mémoire
        delete [] notes;
        // allocation d'un nouvel espace mémoire
        notes=new float[nb_notes];
        // copie des valeurs dans le nouvel espace mémoire
        for(int i=0; i<nb_notes; i++)
            notes[i]=e.notes[i];
    }
    return *this;
}
```

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

```
etudiant.cpp  etudiant.h* x test.cpp
etudiant
class etudiant
{
    int ce;
    string nom;
    int nbNotes;
    float *notes;
    float moyenne;
    int l;
    int c;
    int**mat;
public:
    etudiant& operator=(const etudiant&);
    etudiant(int =999, string = "", int =2);
    etudiant(const etudiant&);
    void saisieNotes();
    void calculMoyenne();
    void afficher(string = "");
    void afficherNotes();
    void remplir();
    void afficherMatiere();
}
```

```
etudiant& etudiant::operator= (etudiant &w)
{
    if(this!=&w) // b=b;
    {
        ce=w.ce;    nom=w.nom;
        nbNotes=w.nbNotes;    moyenne=w.moyenne;
        delete[]notes;
        for(int i=0; i<l; i++)
            delete mat[i];
        delete []mat;
        l=w.l;    c=w.c;
        notes=new float[nbNotes];
        mat=new int*[l];
        for(int i=0; i<l; i++)
            mat[i]=new int[c];
        for(int i=0; i<nbNotes; i++)
            notes[i]=w.notes[i];
        for(int i=0; i<l; i++)
            for(int j=0; j<c; j++)
                mat[i][j]=w.mat[i][j];
    }
    return *this;
}
```

Surcharge des opérateurs <<, >> et = de la classe courbe

```
class courbe
{
    vector<point*> tab;
public:
    courbe();
    void afficher(string = "");
    courbe(const courbe&);
    ~courbe();
    int taille(){return tab.size();}
    void ajouter(point, int =0);
    void ajouter(pointColore, int =0);
    void ajouter(pointColoreMasse, int =0);
    void supprimer(int =0);
    friend ostream& operator<< (ostream&, courbe&);
    friend istream& operator>> (istream&, courbe&);
    courbe& operator= (const courbe&);
};
```

Problème !!!

```
(Global Scope) operator>>(istream & in, courbe & c)
{
    point*q;
    int choix;
    char rep;
    do
    {
        cout<<"\n taper 1:point; 2: pointColore; 3: pointColoreMasse "<<endl;
        in>>choix;
        if (choix==1) q=new point;
        else if (choix==2) q= new pointColore;
        else if (choix ==3) q= new pointColoreMasse;
        else continue;
        in>>*q; // appel operator>> point
        c.tab.push_back(q);
        cout<<"\n rajouter? "<<endl;
        in>>rep;
    }
    while(rep=='o' || rep=='O');    return in; }
```

```
courbe.cpp* x courbe.h test.cpp
(Global Scope) operator<<(ostream & out, courbe & w)

istream& operator>> (istream& in, courbe& w)
{
    int choix;
    char rep;
    do
    {
        cout<<"\n taper 1: point, 2: pointColore, 3: pointColoreMasse"<<endl;
        in>>choix;
        if(choix==1)
        { point* q=new point();
          in>>*q;
          w.tab.push_back(q);
        }
        else if (choix==2)
        {
            pointColore*q=new pointColore();
            in>>*q;
            w.tab.push_back(q);
        }
    }
}
```

Surcharge de l'opérateur >>
de la classe courbe

```
courbe.cpp* x courbe.h test.cpp
(Global Scope) operator>>(istream & in, courbe& w)

    else if(choix==3)
    {
        pointColoreMasse*q=new pointColoreMasse();
        in>>*q;
        w.tab.push_back(q);
    }

    cout<<"\n ajouter ? "<<endl;
    in>>rep;
    }
    while(rep=='o' || rep=='O');

    return in;
}
```

```
istream& operator>> (istream& in, courbe& c)
```

```
{
```

```
    int nb,t;
```

```
    cout<<"\n saisir nbre d'objets "<<endl;
```

```
    in>>nb;
```

```
    for(int i=0; i<nb;i++)
```

```
    {
```

```
        cout<<"\n taper 1: point; 2: pointColore, 3: pointColoreMasse "<<endl;
```

```
        in>>t;
```

```
        if(t==1)
```

```
        {
```

```
            point *q=new point;
```

```
            in>>*q; // appel operator>> de point
```

```
            c.tab.push_back(q);
```

```
        }
```

```
    else if(t==2)
```

```
    {
```

```
        pointColore *q=new pointColore;
```

```
        in>>*q; // appel operator>> de pointColore
```

```
        c.tab.push_back(q);
```

```
    }
```

```
    else if(t==3)
```

```
    {
```

```
        pointColoreMasse *q=new pointColoreMasse;
```

```
        in>>*q; // appel operator>> de pointColoreMasse
```

```
        c.tab.push_back(q);
```

```
    }
```

```
    else break;
```

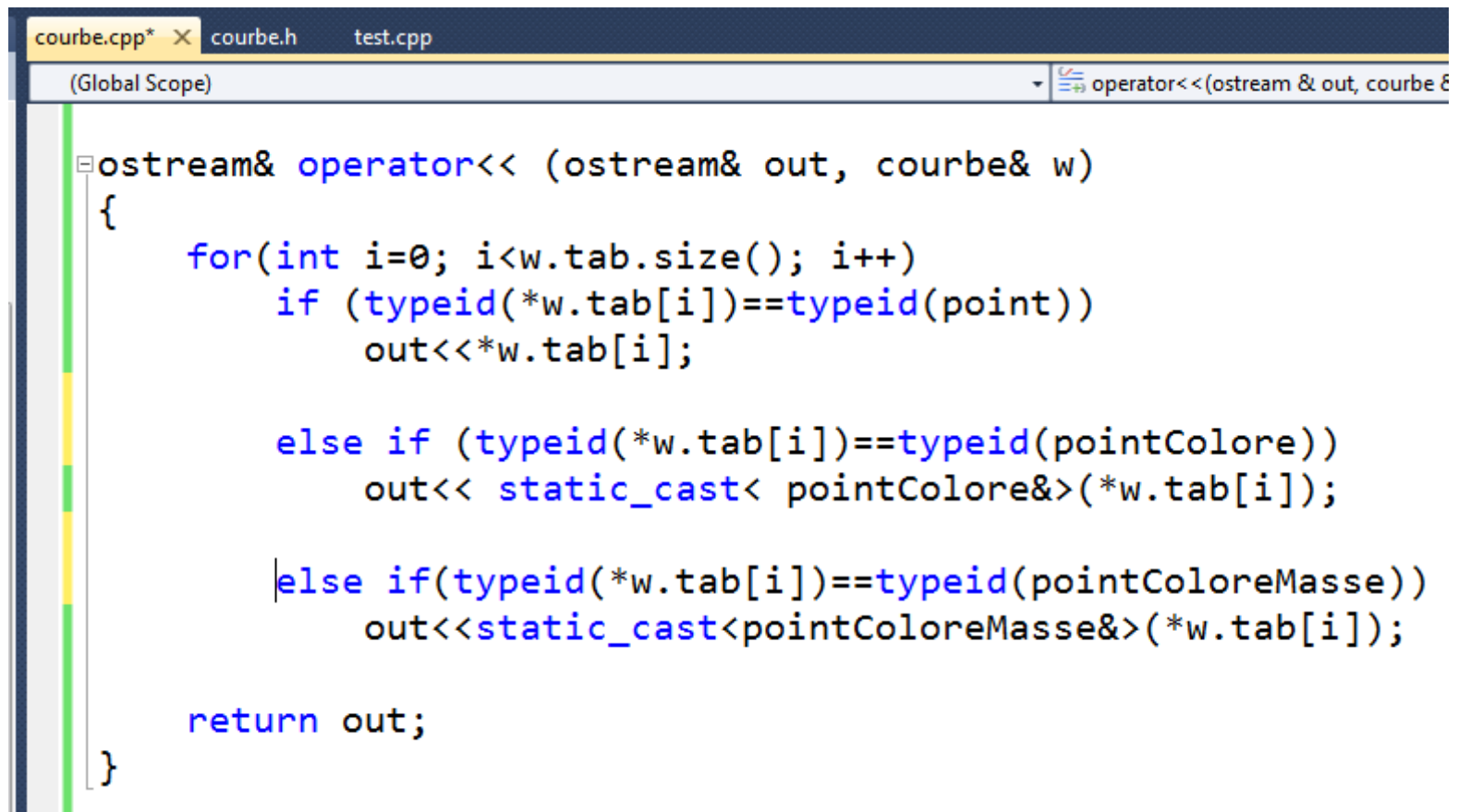
```
}
```

```
return in;
```

```
}
```

Surcharge de
l'opérateur >> de la
classe courbe (version2)

Surcharge de l'opérateur << de la classe courbe



```
courbe.cpp* x courbe.h test.cpp
(Global Scope) operator<<(ostream & out, courbe &
ostream& operator<< (ostream& out, courbe& w)
{
    for(int i=0; i<w.tab.size(); i++)
        if (typeid(*w.tab[i])==typeid(point))
            out<<*w.tab[i];

        else if (typeid(*w.tab[i])==typeid(pointColore))
            out<< static_cast< pointColore&>(*w.tab[i]);

        else if (typeid(*w.tab[i])==typeid(pointColoreMasse))
            out<<static_cast<pointColoreMasse&>(*w.tab[i]);

    return out;
}
```

Surcharge de l'opérateur = de la classe courbe

```
courbe& courbe::operator= (const courbe& w)
{
    if(this!=&w)
    { // libération de l'ancien espace mémoire
        for(int i=0; i<tab.size(); i++)
            delete tab[i];
        tab.clear();
        // duplication des objets
        point *q;
        for(int i=0; i<w.tab.size(); i++)
        {
            if (typeid(*w.tab[i])==typeid(point))
                q=new point(*w.tab[i]);
            else if(typeid(*w.tab[i])==typeid(pointColore))
                q=new pointColore( static_cast<const pointColore&>(*w.tab[i]));
            else if(typeid(*w.tab[i])==typeid(pointColoreMasse))
                q=new pointColoreMasse( static_cast<const pointColoreMasse&>(*w.tab[i]));
            tab.push_back(q); } }
    return *this; }
```

Forme canonique de coplien d'une classe

- La forme canonique, dite de *Coplien*, fournit un cadre de base à respecter pour les classes dont certains attributs sont alloués dynamiquement.

```
class T
{
    public:
        T(.....); // constructeur pour l'allocation des parties dynamiques de l'objet
        T(const T&); // constructeur de copie
        ~T(); //destructeur pour la libération des parties dynamiques de l'objet
        T& operator= (const T &); // affectation
        .....
};
```

Surcharge de l'opérateur d'affectation dans une classe dérivée

```
etudiant_salarie.h*  etudiant.h  main.cpp*  etudiant.cpp  etudiant_salarie.cpp*
(Global Scope)
#pragma once
#include <iostream>
using namespace std ;
class etudiant
{
protected:
    int nb_notes;
    int *notes;
public:
    etudiant(int =2);
    void remplir();
    etudiant(const etudiant &);
    void afficher(char* = "");
    etudiant& operator= (const etudiant&);
    ~etudiant(void);
    int operator[] (int);
    friend ostream& operator<< (ostream&, etudiant&);
    friend istream& operator>> (istream&, etudiant&);
};
```

```
etudiant_salarie.h*  etudiant.h  main.cpp*  etudiant.cpp*  etudiant_salarie.cpp*
(Global Scope)
#pragma once
#include "etudiant.h"
class etudiant_salarie: public etudiant
{
    int nb_mois;
    int *mois;
public:
    etudiant_salarie(int =2, int =3);
    void remplir();
    void afficher(char* = "");
    etudiant_salarie(const etudiant_salarie &);
    ~etudiant_salarie(void);
    etudiant_salarie& operator= ( etudiant_salarie&);
    int operator[] (int);
    friend ostream& operator<< (ostream&, etudiant_salarie&);
    friend istream& operator>> (istream&, etudiant_salarie&);
};
```

```

etudiant& etudiant::operator=(const etudiant &w)
{
    cout<<"\n DEBUT SURDEFINITION = etudiant "<<endl;
    if(this !=&w)
    {
        delete[] notes;
        nb_notes=w.nb_notes;
        notes=new int[nb_notes];
        for(unsigned int i=0; i<nb_notes; i++)
            notes[i]=w.notes[i];
    }
    cout<<"\n FIN SURDEFINITION = etudiant"<<endl;
    return *this;
}

```

```

etudiant_salarie& etudiant_salarie::operator= ( etudiant_salarie &w)
{
    cout<<"\n DEBUT SURDEFINITION = etudiant salarie "<<endl;
    if (this !=&w)
    {
        etudiant *ad1=this;
        etudiant *ad2=&w;
        // conversion des pointeurs sur etudiant_salarie this et &w
        // en des pointeurs sur etudiant
        *ad1= *ad2;
        // cette affectation entraine l'appel de l'operateur d'affectation
        // de la classe etudiant
        delete[] mois;
        nb_mois=w.nb_mois;
        mois=new int [nb_mois];
        for(unsigned int i=0; i<nb_mois; i++)
            mois[i]=w.mois[i];
    }
    cout<<"\n FIN SURDEFINITION = etudiant salarie "<<endl;
    return *this;
}

```

```
ostream& operator<< (ostream& sortie, etudiant_salarie &es)
{
    cout<<"\n DEBUT SURDEFINITION << etudiant_salarie"<<endl;
    etudiant*e=&es;
    sortie<<*e;
    sortie<<"\n le nombre de mois est: "<<es.nb_mois;
    sortie<<"\n Les mois sont: "<<endl;
    for(unsigned int i=0; i<es.nb_mois; i++)
        sortie<<es[i]<<"\t";
    cout<<endl;
    cout<<"\n FIN SURDEFINITION << etudiant_salarie "<<endl;
    return sortie;
}
```

*Appel de operator<< de la classe etudiant pour afficher
nb_notes et les notes*

```
istream& operator>> (istream& entree, etudiant_salarie& es)
{
    cout<<"\n DEBUT SURDEFINITION >> etudiant_salarie "<<endl;
    etudiant*e = &es;
    entree>>*e;
    cout<<"\n saisir le nbre de mois: "<<endl;
    for(unsigned int i=0; i<es.nb_mois; i++)
        entree>>es.mois[i];
    cout<<"\n FIN SURDEFINITION >> etudiant_salarie "<<endl;
    return entree;
}
```

*cout<<"\n saisir le nbre de mois "<<endl;
entree>>es.nb_mois;*

Appel de operator>> de la classe etudiant pour saisir nb_notes et les notes

<pre> class T { public: T (...); // constructeurs T (const T &) ; // constructeur de copie de T ~T () ; // destructeur T & operator = (const T &) ; // opérateur d'affectation ... }; </pre>	<div>Formes canoniques de la classe de base T et de la classe dérivée U</div> <ul style="list-style-type: none"> • U est sous classe de T • T et U possèdent chacune des parties dynamiques
<pre> class U : public T { public: U (...); // constructeurs U (const U & x) : T (x) // constructeur copie de U : utilise celui de T { prévoir ici la copie de la partie spécifique à U } ~U(); U & operator= (U & w) //opérateur d'affectation { T * ad1 = this; T * ad2 = &w ; *ad1 = *ad2 ; // affectation de la partie spécifique à T // prévoir ici l'affectation de la partie spécifique à U } } </pre>	