

CONSTRUCTEURS & HERITAGES

HEADER

```
class Point
{
    public :
    float x,y;
    // constructeur par défaut
    point();
    // constructeur paramétrique
    point(float _x, float _y);
    ~Point();
};

class PointC : public Point
{
    public :
    int Couleur;
    pointC();
    pointC(float _x, float _y, int _C);
    ~PointC();
};
```

chaînage des destructeurs automatique

SOURCE

```
// constructeur par défaut
point::point() {}
point::point() { x=0; y=0;}

// constructeur paramétrique
syntaxe 1
point::point(float _x,float _y)
{ x=_x; y=_y; }

syntaxe 2
point::point(float _x,float _y)
x(_x), y(_y) { }

// chaînage des constructeurs
syntaxe 1
pointC::pointC(float _x,float _y, int _C) :
point(_x,_y)
{ couleur = _C; }

syntaxe 2
pointC::pointC(float _x,float _y, int _C) :
point(_x,_y),
couleur(_C) { }

syntaxe 3
pointC::pointC(float _x,float _y, int _C)
{ x=_x; y=_y; couleur = _C; }
```



appel du constructeur par défaut de Point

Lilian BUZER ESIEE
Dept. Informatique
Fiche rappel langage C/C++

UTILISATION

```
Point P;
Point D(3,4);
Point A = P;
Point C = P(4,5); → Point C(4,5);
P = C;
Point L[100];
Point * LL = new Point[200];
// si opérateur fourni
P = D+C+P(4,8);
```

POLYMORPHISME D'HERITAGE

```
class A
{
    public :
    virtual Aff() { cout << « A »; }
};

class B : public A
{
    public :
    Aff() { cout << « B »; }
};

A T[2]; T[0] = A(); T[1] = B();
T[0].Aff(); → A
T[1].Aff(); → A (problème)

A * T[2]; T[0] = new A(); T[1] = new B();
T[0]->Aff(); → A
T[1]->Aff(); → B (correct)
```