LA NOTION DE SURCHARGE

Méthodes et constructeurs sont identifiés par leur nom, la liste des paramètres passés en arguments et éventuellement la valeur retournée par la fonction.
La liste des arguments est appelée « **signature** » de la fonction.
La surcharge des méthodes, opérateurs et constructeurs consiste à implémenter dans une même classe, plusieurs fonctions de mêmes noms mais de signatures différentes.
Ce mécanisme s'applique aussi au opérateurs et constructeurs.

Danger :
Lorsque l'on utilise des fonctions de signatures différentes avec valeurs pré initialisées (notamment pour les constructeurs), le compilateur provoquera une erreur car il ne sera pas capable de choisir quelle fonction implanter.

EX :

```
Class   truc
        {
        private :
        int a ;
        public :
        truc ( ) ;                              //constructeur sans paramètre
        truc ( int dd , char * machin ) ;       //constructeur avec paramètre
        //truc ( int dd = 0 , char * machin = « classe truc » ) ;//constructeur avec paramètres pré
initialisés.
        add ( int arg1 , int arg2) ;            // this.a  = arg1  +  arg2
        int  add ( int arg1 ) ;             //valeur retournée = arg1 + this.a
        } ;
```

EX :

```
#define PI  M_PI
class nb_cmpx
{
private :
void calcul();

public :
float preel;
float pimag;
float  module;
float  angle;
nb_cmpx(float init_preel, float init_pimag);
// nb_cmpx(float init_preel = 0, float init_pimag = 0);//ambigu s'il existe
// nb_cmpx () ;
void add (nb_cmpx n1 , nb_cmpx n2);//this = n1 + n2
void visu();
nb_cmpx operator + (nb_cmpx n1);
};
nb_cmpx::nb_cmpx(float init_preel =0, float init_pimag =0)
{
preel = init_preel;
pimag = init_pimag;
calcul();
}
void nb_cmpx::add (nb_cmpx n1 , nb_cmpx n2)
{
preel = n1.preel + n2.preel;
pimag = n1.pimag + n2.pimag;
calcul();
}
void nb_cmpx::calcul()
{
module = hypot (preel , pimag);
angle =  (atan2(pimag , preel))*360/2/PI;
}
void nb_cmpx::visu()
{
cout << "\nNombre complexe :";
cout <<"\np reelle\t: "<<preel<<"\tpimag\t: "<<pimag;
cout <<"\nmodule\t: "<<module<<"\targument\t: "<<angle;
}
nb_cmpx nb_cmpx::operator + (nb_cmpx n1)
{
nb_cmpx n;
n.preel = preel + n1.preel;
n.pimag = pimag + n1.pimag;
n.calcul();
return (n);
}
int main()
{
nb_cmpx n1(-1,1),n2(1,1);
nb_cmpx n3;
//n3.add(n1,n2);
n3 = n1 + n2;
n3.visu();
char *suite = (char *)malloc(10);
cin >> suite;
        return 0;
}
```