
Chapitre 9

Constructeur de copie, affectation

Le clonage d'objets : recopie et affectation

- ✓ Recopier un objet dans un autre est une opération assez fréquente.
- ✓ Deux fonctionnalités y sont dédiées en C++ :
le constructeur par recopie et l'opérateur d'affectation.

1- Le constructeur par recopie

1.1- Motivation

- ✓ Le constructeur par recopie est très important car il permet d'initialiser un objet par clonage d'un autre.
- ✓ Ce qui signifie que l'objet est en cours de construction.
- ✓ En particulier, le constructeur par recopie est invoqué dès lors que l'on passe un objet par valeur à une fonction ou une méthode.

1.2- Syntaxe

✓ La syntaxe du constructeur par copie d'une classe T est la suivante :

T::T(const T& o);

- ✓ Il est extrêmement important de passer l'objet copié par référence sous peine d'entraîner un appel récursif infini !

1.3- Que se passe t'il si vous ne fournissez pas de constructeur de copie ?

✓ Si vous ne fournissez pas explicitement de constructeur par copie, le compilateur en génère automatiquement un pour vous.

✓ Celui-ci effectue une copie binaire optimisée de votre objet, ce qui est parfait si celui-ci ne contient que des éléments simples.

✓ En revanche, si votre objet contient des pointeurs, ce sont les valeurs des pointeurs qui vont être copiées et non pas les variables pointées, ce qui dans de nombreux cas, conduira directement à la catastrophe.

1.4- Quand doit on fournir un constructeur de copie ?

✓ Vous devez fournir un constructeur de copie dès lors que le clonage d'un objet par copie binaire brute peut entraîner un dysfonctionnement de votre classe, c'est à dire, en particulier :

- Utilisation de mémoire dynamique
- Utilisation de ressources systèmes (fichiers, sockets, etc ...)

2- L'opérateur d'affectation

2.1- Mise en place

✓ L'opérateur d'affectation et le constructeur de copie sont très proches dans le sens où ils sont requis dans les mêmes circonstances et qu'ils effectuent la même opération : cloner un objet dans un autre.

✓ Il y a tout de même une différence fondamentale :

- L'opérateur d'affectation écrase le contenu d'un objet déjà existant et donc totalement construit.

- Ce qui signifie que dans la majorité des cas, il faudra commencer par "nettoyer" l'objet à la manière d'un destructeur avant d'effectuer l'opération de clonage dessus.

2.2- Syntaxe

Le prototype de l'opérateur d'affectation d'une classe Test le suivant :

T & operator=(const T& o);

3- Initialisation ou Affectation

✓ Il est parfois délicat de savoir si l'on a affaire à une affectation ou une initialisation car la syntaxe du signe "=" peut être trompeuse.

✓ Il existe pourtant une règle simple :

Toute opération d'initialisation ou d'affectation dans une déclaration est l'affaire d'un constructeur

✓ Le tableau suivant résume quelques cas:

Instruction	Description	Méthode mise en jeu
T t1;	Initialisation par le constructeur par défaut	T::T(void);
T t2(<i>params</i>);	Initialisation par un constructeur quelconque	T::T(<i>liste params</i>);
T t3(t1);	Initialisation par le constructeur de copie	T::T(const T&);
T t4();	Piège à c... c'est le prototype de la fonction t4 qui ne prend pas de paramètre mais renvoie un objet de type T.	
T t5=t1	Initialisation par le constructeur de copie Cette ligne est à remplacer par T t5(t1); qui fait exactement la même chose mais est moins ambiguë du point de vue de la syntaxe.	T::T(const T&);
t5=t2	Affectation à l'aide de l'opérateur d'affectation	T & T::operator=(const T&);


```

class liste
{
private :
int taille;
float *adr;
public:
liste(int t);
liste(liste &v);
void saisie();
void affiche();
liste oppose();
~liste();
};

```

```

#include <liste.h>
#include <iostream>
Using namespace std,
int main()
{
cout<<"Début de main()"<<endl;
liste a(3), b(3);
a.saisie(); a.affiche();
b = a.oppose(); b.affiche();
liste c=a ;
c. affiche();
cout<<"Fin de main()"<<endl;
return 0;
}

```

```

#include <liste.h>
#include <iostream>
Using namespace std,

liste::liste(int t)
{
taille = t; adr = new float[taille];
cout<<"Construction";
cout<<" Adresse de l'objet:"<<this;
cout<<" Adresse de liste:"<<adr<<endl;
}

liste::liste(liste &v) // passage par référence obligatoire
{
taille = v.taille;
adr = new float[taille];
for(int i=0;i<taille;i++)adr[i]=v.adr[i];
cout<<"Constructeur par recopie";
cout<<" Adresse de l'objet:"<< this;
cout<<" Adresse de liste:"<< adr <<endl;
}

liste::~~liste()
{
cout<<"Destruction Adresse de l'objet:"<< this;
cout<<" Adresse de liste:"<< adr <<endl; delete adr;
}

void liste::saisie()
{
for(int i=0;i<taille;i++)
{
cout<<"Entrer un nombre:"; cin>>*(adr+i);
}
}

void liste::affiche()
{
for(int i=0;i<taille;i++)cout<<*(adr+i)<<" ";
cout<<"adresse de l'objet: "<< this;
cout<<" adresse de liste: "<< adr <<endl;
}

liste liste::oppose()
{
liste res(taille);
for(int i=0;i<taille;i++)res.adr[i] = - adr[i];
for(i=0;i<taille;i++)cout<<res.adr[i]<<" ";
cout<<endl;
return res;
}

```