Extrait du référentiel : BTS Systèmes Numériques option A (Informatique et Réseaux)		Niveau(x)
S4. Développement logiciel		
S4.6. Programmation orientée objet (Support : C++)	Définition de classes (encapsulation) et modèle canonique (dit de Coplien)	3
	Surcharges d'opérateurs (injection, etc.)	3
S4.7. Langages de programmation	C++	3

Objectifs du cours :

- Constructeur de recopie :
  - constructeur de recopie implicite
  - constructeur de recopie explicite

CONSTRUCTEUR DE RECOPIE

Dans l'activité pratique sur le langage C++ : « exercice sur les appels de constructeurs », vous avez été confronté aux lignes de code suivante :

```
CSosie objet3(m, y);
CSosie objet11 = objet3; // Création et initialisation d'un objet avec un objet existant
```

Vous aviez constaté que l'affectation (la recopie des attributs) avait été réalisée, sans pour autant connaître le constructeur qui avait été appelé.
Que s'était-il passé ?
Il y a eu appel d'un autre constructeur appelé le **constructeur de recopie implicite** pour `objet11`.
Un objet peut-être transmis en argument à une fonction ou à une méthode (avec passage de l'argument par valeur). Dans ce cas, il se crée dans un emplacement local à la méthode ou à la fonction, une copie de l'objet transmis (c'est l'objet temporaire).

Le cas se présente aussi lorsqu'un argument de type classe est retourné par valeur comme résultat d'une fonction ou d'une méthode.

Une troisième situation, analogue aux deux précédentes, peut se produire : le cas où un objet est initialisé lors de sa création avec un autre objet du même type, c'est une situation de recopie également.

Dans ces trois cas, il y a **initialisation par recopie**. Cela consiste à la création d'un objet par recopie d'un objet existant de même type.

Si on ne prend pas quelques précautions lors de la recopie, il s'avère que sur une copie « classique » ou « superficielle », seules les valeurs des pointeurs seront copiées, les emplacements pointés ne le seront pas.
Le langage C++ a prévu d'utiliser un constructeur par recopie pour palier à ce problème.

Dans tous les cas d'initialisation par recopie, il y a toujours appel d'un constructeur de recopie :
- soit ce dernier est implicite : c'est **le constructeur de recopie implicite**. Dans ce cas la recopie est superficielle.
- soit ce dernier est explicite : on peut personnaliser la recopie, allouer un nouvel emplacement mémoire.

Le constructeur de recopie se reconnaît par sa déclaration. Il prend un seul argument du type de sa classe. Le passage se fait par référence obligatoirement :

`point(point &)` ou **`point(const point &)`**

Toutes les notions présentées dans le cours sur les constructeurs sont bien évidemment applicables pour les constructeurs de recopies (déclaration, surdéfinition. …).
On parlera de **constructeurs usuels** (pour les constructeurs que l'on utilise habituellement) pour les différencier des constructeurs de recopie.

CONSTRUCTEUR DE RECOPIE IMPLICITE

Exemple : une classe **`Ctab`** permettant de gérer des tableaux d'entiers de taille variable.

```
#include <iostream>

using namespace std;

class Ctab
{
private:
 int nelem; // nombre d'elements
 double *adr; // pointeur sur ces elements

public:
 Ctab(int); // constructeur usuel à un argument
 ~Ctab();
};

Ctab::Ctab(int n)
{
 adr = new double[nelem = n];
 cout << "Constructeur usuel - adresse objet : " << this << endl
```

```
       << "Adresse des elements pointes : " << adr << endl;
}
Ctab::~Ctab()
{
 cout << "Destruction de l objet - adresse objet : " << this << endl
       << "Destruction du vecteur d adresse : " << adr << endl;
 delete adr;
}
void fct(Ctab b) // affectation sans constructeur de recopie
{
 cout << "***** Appel de la fonction fct *****\n";
}

int main()
{
 Ctab a(5);
 fct(a); // objet transmis en argument à une fonction
 return 0;
}
```

Le type objet tableau n'existe pas en langage C++. Pourquoi ne pas le créer !

Cet exemple permet de créer des objets « tableau » de taille personnalisée. Le constructeur usuel reçoit en argument la taille souhaitée du tableau puis il réalise une allocation dynamique pour la réserver.

Dans la fonction principale, un objet tableau **a** est créé puis transmis par valeur à une fonction ordinaire nommée **fct()**. L'appel de **fct(a)** crée un nouvel objet **b**.
On a recopié dans ce nouvel objet les valeurs des membres données de l'objet **a** (passage d'argument de type classe par valeur).

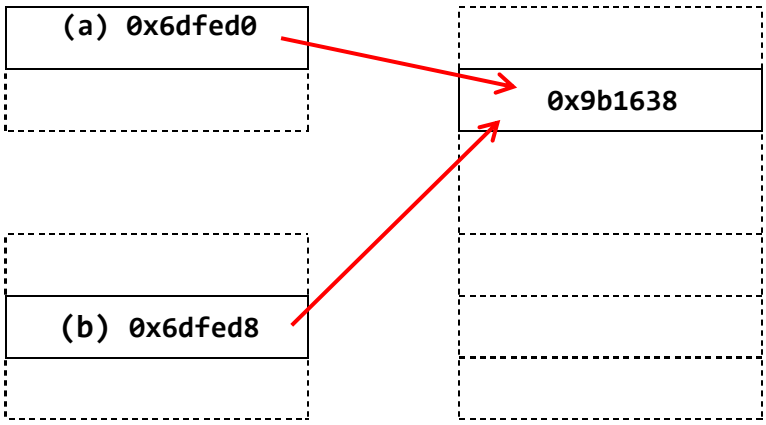
Résultat :

```
Constructeur usuel - adresse objet : 0x6dfed0
Adresse des elements pointes : 0x9b1638
***** Appel de la fonction fct *****
Destruction de l objet - adresse objet : 0x6dfed8
Destruction du vecteur d adresse : 0x9b1638
Destruction de l objet - adresse objet : 0x6dfed0
Destruction du vecteur d adresse : 0x9b1638
```

On a réalisé :

```
Ctab b = a; // création de l'objet b et initialisation de ce dernier par recopie de a
// création de l'objet temporaire
```

La situation peut être schématisée par un plan mémoire :



On constate qu'à la fin de l'exécution de la fonction **fct()**, il y a appel du destructeur **~Ctab()**, pour l'objet **b**, ce qui libère l'emplacement pointé par **adr**, puis à la fin de l'exécution de la fonction **main()**, il y a de nouveau appel du destructeur **~Ctab()** pour l'objet **a**, ce qui libère …
le même emplacement mémoire : SURPRENANT !

Cette opération constitue une erreur d'exécution dont les conséquences peuvent être graves. Imaginez qu'après la libération de cet espace mémoire par l'objet **b**, cet espace est attribué à d'autres variables par l'ordinateur, et ensuite lorsque l'on quitte la fonction **main()**, ce même espace mémoire est à nouveau effacé par l'objet **a** !

Pour éviter ce problème, nous devons faire en sorte que l'appel **fct(a)** conduise à créer un nouvel objet avec ses données membres mais surtout avec son propre emplacement mémoire de stockage.

Pour cela, nous devons ajouter un **constructeur de recopie explicite** dans notre programme qui va permettre de personnaliser et de sécuriser l'application.

CONSTRUCTEUR DE RECOPIE EXPLICITE

La déclaration de ce constructeur de recopie explicite est :

```
Ctab(Ctab &) // passage par référence
```

Ce constructeur doit :

Créer dynamiquement un nouvel emplacement dans lequel il recopie les valeurs correspondant à l'objet reçu en argument.
Renseigner convenablement les membres données du nouvel objet.

Le programme précédent modifié :

```
#include <iostream>

using namespace std;

class Ctab
```

```
{
private:
 int nelem; // nombre d'elements
 double *adr; // pointeur sur ces elements

public:
 Ctab(int); // constructeur usuel à un argument
 Ctab(Ctab &); // constructeur de recopie
 ~Ctab();
};

Ctab::Ctab(int n)
{
 adr = new double[nelem = n];
 cout << "Constructeur usuel - adresse objet : " << this << endl
      << "Adresse des elements pointes : " << adr << endl;
}

Ctab::Ctab(Ctab &v) // constructeur explicite de recopie
{
 adr = new double[nelem = v.nelem]; // création du nouvel objet
 for (int i = 0; i < nelem; i++)
      adr[i] = v.adr[i]; // recopie de l'ancien objet
 cout << "Constructeur de recopie - adresse objet : " << this << endl
      << "Adresse des elements pointes : " << adr << endl;
}

Ctab::~Ctab()
{
 cout << "Destruction de l objet - adresse objet : " << this << endl
      << "Destruction du vecteur d adresse : " << adr << endl;
 delete adr;
}

void fct(Ctab b) // affectation sans constructeur de recopie
{
 cout << "***** Appel de la fonction fct *****\n";
}

int main()
{
 Ctab a(5);
 fct(a); // objet transmis en argument à une fonction
 return 0;
}
```

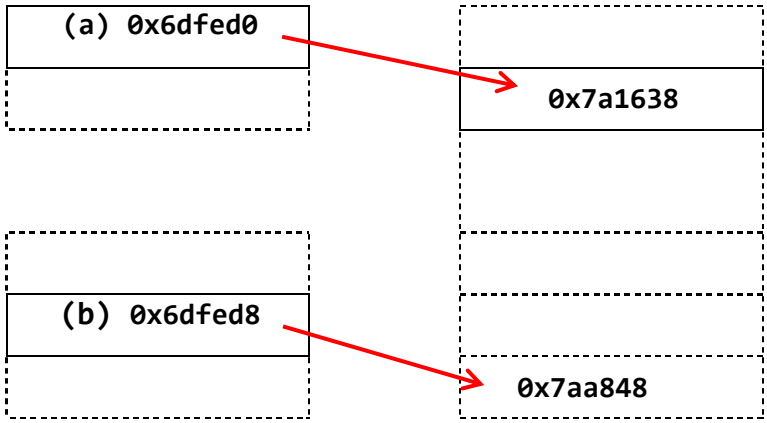
Résultat :

```
Constructeur usuel - adresse objet : 0x6dfed0
Adresse des elements pointes : 0x7a1638
Constructeur de recopie - adresse objet : 0x6dfed8
```

```
Adresse des elements pointes : 0x7aa848
***** Appel de la fonction fct *****
Destruction de l objet - adresse objet : 0x6dfed8
Destruction du vecteur d adresse : 0x7aa848
Destruction de l objet - adresse objet : 0x6dfed0
Destruction du vecteur d adresse : 0x7a1638
```

On constate cette fois que chaque objet possède son propre emplacement mémoire, et que les destructions successives se déroulent sans problème.

Le nouveau plan mémoire :



Les deux règles :

- le constructeur de recopie doit être appelé lors de l'initialisation (ou affectation) d'un objet existant sur un objet nouvellement créé.
- l'emploi d'un constructeur de recopie est obligatoire dès que l'un des attributs de l'objet est dynamique (pointeur).

À votre avis, est-ce que le constructeur de recopie peut être utilisé dans la situation suivante ?

```
Ctab a(5);
Ctab b(6);
a = b; // affectation d'objet, les deux objets sont déjà existants !
```

Réponse :

……

……

……