Constructeur par copie (recopie) (clonage)

Rappel :

Un constructeur est une fonction qui est appelée **automatiquement** lors de la **création** d'un objet.
- **un** : il peut y en avoir plusieurs pour une même classe,
- **automatiquement** : on ne peut pas l'appeler explicitement comme les autres fonctions,
- **création** : un objet est créé par exemple lors de sa déclaration, de son allocation, mais aussi de manière implicite (invisible) lors du passage d'un objet à une fonction par valeur ou lors du retour d'un objet d'une fonction par valeur,

Si il peut y avoir plusieurs constructeurs pour une classe, leur distinction se fait par les arguments qu'ils possèdent, et on peut se servir de ces arguments pour choisir d'initialiser les données membres des objets.

Il existe un constructeur particulier qui est celui qui reçoit en argument un objet de la classe, exemple avec la classe Point :

```
Point::Point(Point & P)
```

Notez le passage par référence, on y reviendra plus tard.

Le contenu de ce constructeur est simple, l'idée est de recopier l'objet en argument dans l'objet implicite créé, par exemple :

```
{
  x = P.x ;
  y = P.y ;
}
```

Sachant que si une classe ne possède pas de constructeur de copie, le compilateur en génère un par défaut qui va recopier tous les membres de l'objet.
L'intérêt d'écrire son constructeur est essentiel dans les cas où un membre (ou plusieurs) est dynamique (alloué avec new) car dans ce cas, ce n'est pas l'espace qui est recopié mais seulement le pointeur et les 2 objets se retrouvent à pointer sur le même espace. C'est rarement l'effet attendu.

Le risque est encore plus important si on a un destructeur de l'objet :

Par exemple si il s'agit d'une copie générée lors d'un passage de paramètre par valeur à une fonction, l'objet va disparaître après la fonction et l'espace mémoire ne sera plus disponible dans l'objet initial.

Boucle sans fin.

Attention à l'en-tête de ce constructeur, l'argument doit être passé par référence.

`Point::Point(Point & P)`

Et pas :

```
Point::Point(Point P)
```

Si l'argument n'est pas passé par référence, c'est à dire qu'il est passé par valeur. Or une argument passé par valeur est une variable locale initialisée par la valeur de l'argument effectif.

Une variable locale est donc créée, ce qui provoque l'appel d'un constructeur, mais quel constructeur ?

Le constructeur qui a pour argument un objet du même type, c'est à dire le constructeur par copie !

D'où il se produit une boucle récursive sans point d'arrêt qui génère un plantage du programme !