La sémantique de déplacement en C++ 11

Lors de ce Workshop, nous allons aborder une des fonctionnalités emblématiques du C++ moderne : “la sémantique de déplacement”.

Premier exercice :

Dans ce fichier se trouve un véritable souci d’optimisation, à chaque fois que la fonction **"doSomethingWithMyBigClass"** est utilisée, le mauvais constructeur est appelé, mais celui-ci est long, pour utiliser la valeur, il a besoin de TOUT copier.

Comment remédier à cela ?

Contrainte : Seul la ligne **47** de l’exercice est modifiable.

But de l’exercice : Optimisez le programme et faites-en sorte qu’il appelle le bon constructeur.

Output attendu :

```
± % ./exo1
MyBigClass constructed !
MyBigClass moved !
Something has been done !
MyBigClass destroyed !
MyBigClass destroyed !
```

Deuxième exercice :

Dans le premier exercice, nous avons créé des faux constructeurs, il est temps désormais de les remplir.

Attention, le constructeur de déplacement et de copie n'ont pas le même but.

But de l'exercice : Remplissez les deux constructeurs de sorte à ce que le programme ne crash pas.

Output attendu :

```
± % ./exo2
initial array
420 420 420 420 420 420 420 420 420 420
copied array
69 69 69 69 69 69 69 69 69 69
moved array
42 42 42
```

Troisième exercice :

Dans ce bout de code, une rvalue est passée à une fonction attendant une référence rvalue.

Dans cette sous fonction, nous envoyant notre référence rvalue à une autre fonction attendant une référence rvalue.

Pourquoi cela ne fonctionne pas ?

But de l'exercice : Faites-en sorte que cet exercice compile.

Output attendu :

```
axel@Axel-ubuntu:~/CLionProjects/movesemantic/WorkshopMoveCXX$ ./exo3
je fais des trucs
Les coordonées sont 143 4
axel@Axel-ubuntu:~/CLionProjects/movesemantic/WorkshopMoveCXX$
```

Quatrième exercice :

Nous avons vu jusqu'à présent les références rvalue et les référence lvalue. Nous avons vu aussi lors de la présentation que différents types de "values" héritent de celles-ci.

Cependant, il reste un dernier type de "value", un type dit **universel**.

But de l'exercice : La fonction appelée appelle une autre fonction polymorphique, cependant, une seule fonction semble appelée quel que soit le type de value. Faites-en sorte que la bonne fonction soit appelée.

Output attendu :

```
axel@Axel-ubuntu:~/CLionProjects/movesemantic/WorkshopMoveCXX$ ./exo4
lvalue detected
rvalue
You won !
axel@Axel-ubuntu:~/CLionProjects/movesemantic/WorkshopMoveCXX$
```