`const` et `constexpr`

J.-C. Chappelier, J. Sam

version 1.0 de septembre 2016

La norme C++11 a introduit le mot réservé « `constexpr` » qui est sensiblement différent du mot réservé « `const` », mais leur similarité est parfois source de confusions. Essayons de clarifier leurs emplois.

1 `const` ne veut pas dire « constant »

Pour rappel (cours), le mot réservé « `const` » qualifie un nom de variable [1] pour indiquer qu'*au travers de **ce** nom*, la valeur ne peut pas être modifiée.

Par exemple :

```
int const i(3);
```

empêche de modifier la valeur de `i` : le code suivant ne compilera pas :

```
i = 5;
```

Pourquoi avoir précisé « *au travers de **ce** nom* » ?

Cette nuance fait appel à des notions présentées en fin de cours [2] : les pointeurs et les références. Donnons juste un exemple ici :

```
int const i(3);
int* ptr(&i);
```

`ptr` pointe sur `i`, mais sans être lui-même `const`.
(Attention ! C'est justement de la *mauvaise* programmation !... mais c'est possible [3].)

1. C'est le seul usage vu jusqu'à présent dans ce cours et donc le seul discuté ici. Les autres usage de `const` seront discutés dans le cours « Introduction à la programmation orientée-objet ».

2. Ne vous y attardez donc pas si vous lisez ce complément avant d'avoir vu le cours sur les pointeurs et références. Vous pourrez toujours y revenir le moment voulu.

3. C'est tellement de la mauvaise programmation que certains compilateurs n'accepteront pas de compiler un tel code à moins d'ajouter des options, comme par exemple `-fpermissive` aux dernières versions de `g++`

Alors

```
i = 5;
```

ne compilera toujours pas, mais par contre

```
*ptr = 5;
```

est tout à fait possible et changera la valeur de `i` ! Le `const` sur `i` ne veut donc pas dire que la valeur de `i` ne peut pas changer dans l'absolu, mais bien qu'elle ne peut pas changer *au travers du nom* `i`.

`const` ne veut donc pas dire « constant », mais « en lecture seule » (sous-entendu, « pas en écriture/affectation »).

2 Une valeur `const` n'est pas nécessairement connue à la compilation

Le qualificatif « `const` » est une notion « *dynamique* (*runtime*) » en ce sens qu'il peut s'appliquer à des valeurs non connues au moment de la compilation (« *compile-time* »).

Prenons un exemple : supposons que l'on demande une valeur `i` à l'utilisateur de notre programme et que, une fois entrée, la valeur de `i` ne sera plus modifiée dans la suite. On pourrait bien sûr écrire :

```
int i;
cout << "Entrez une valeur : ";
cin >> i;
```

mais cela ne souligne pas, ne force pas, le fait que la valeur `i` ne soit plus être modifiée dans la suite.

Pour bien marquer cela il serait préférable d'écrire :

```
int lue;
cout << "Entrez une valeur : ";
cin >> lue;
const int i(lue); // i est initialisé avec la valeur lue
```

On voit bien sur cet exemple que :

- une fois donnée, la valeur de `i` ne peut pas être modifiée ;
- pourtant la valeur que prend effectivement `i` n'est pas connue au moment de compiler le programme.

3 `constexpr` signifie « connu à la compilation et constant »

C'est justement pour palier aux deux subtilités du mot réservé `const` pointées par les deux sections précédentes que le comité du C++ a décidé dans la version 2011 d'introduire

un nouveau mot réservé, `constexpr`, qui signifie justement « connu à la compilation et constant ».

Une variable (ou plus largement une expression, mais nous n'aborderons pas cela ici) peut être qualifiée de `constexpr` si *justement* ces deux conditions sont remplies :

— on connait sa valeur au moment de la compilation ;
— cette valeur ne changera pas au cours du programme.

Par exemple :

```
constexpr double pi(3.14159265358979323846);
```

Le fait que la valeur doit être connue au moment de la compilation interdit bien sûr d'utiliser `constexpr` dans l'exemple de la section précédente :

```
int lue;
cout << "Entrez une valeur : ";
cin >> lue;
constexpr int i(lue);
```

Un tel code ne compile pas.

Maintenant que vous avez lu ce complément, nous vous conseillons d'utiliser `constexpr` partout où les deux conditions de son application sont remplies.