

Beste de savoir

## Causerie++ - Episode 5

---

4 mai 2019



# Table des matières

1.	Introduction . . . . .	1
2.	[C++20] Concepts . . . . .	1
2.1.	Quelques ressources . . . . .	3
3.	Conclusion . . . . .	3

## 1. Introduction

Salut les agrumes! Bienvenue dans ce nouvel épisode de Causerie++. Pour les nouveaux, vous pouvez trouver l'épisode précédent [ici](#) .

Cet épisode a une semaine de retard, je m'en excuse. J'étais trop occupé personnellement ces derniers temps, je n'ai pas réussi à le sortir la semaine dernière.

Trêve de mondanités, aujourd'hui, on va enfin parler des Concepts!

## 2. [C++20] Concepts

Les Concepts sont une nouvelle fonctionnalité qui rend l'utilisation de templates beaucoup plus agréable qu'avant. Et personnellement, j'adore les templates, donc ça me plaît beaucoup!

L'idée des concepts est de rendre moins générique les paramètres templates. En effet, en C++, on peut soit déclarer une fonction « classique », soit déclarer une fonction template avec des paramètres complètement génériques. Avec les Concepts, on pourra restreindre les paramètres templates pour qu'ils respectent certaines conditions.

Cela a plusieurs conséquences très agréables.

### 2.0.1. Des erreurs de compilation bien plus claires

On reproche souvent aux templates de générer des erreurs absolument incompréhensibles. Avec les concepts, beaucoup de ces erreurs seront remplacés par de jolis :

```
1 error: cannot call std::sort with std::_List_iterator<int>
2 note: concept RandomAccessIterator<std::_List_iterator<int>> was
   not satisfied
```

Cet exemple est tiré de [cette page](#) .

## 2. [C++20] Concepts

### 2.0.2. Des interfaces plus claires

Ce second point est fortement corrélé au premier. grâce aux Concepts, les conditions que doivent respecter nos paramètres templates ne sont plus implicites ; elles font directement partie de l'interface de nos fonctions templates. Ainsi, en plus d'avoir des erreurs plus claires, les interfaces elles-mêmes deviennent limpides.

### 2.0.3. Simplification de spécialisations

Grâce aux Concepts, on pourra simplifier des spécialisations nécessitant de faire du SFINAE en les remplaçant par de simples surcharges basées sur des Concepts. En plus de rendre le code plus facile à écrire, cela le rend surtout beaucoup plus expressif.

Franchement, qui préfère ceci :

```
1  template<typename IterT>
2  IterT increment_impl(IterT iter, int n, std::forward_iterator_tag)
3  {
4      assert(n >= 0);
5      for(int i = 0; i < n; ++i)
6          ++iter
7
8      return iter;
9  }
10
11 template<typename IterT>
12 IterT increment_impl(IterT iter, int n,
13                     std::random_access_iterator_tag)
14 {
15     return iter + n;
16 }
17 template<typename IterT>
18 IterT increment(IterT iter, int n)
19 {
20     return increment_impl(iter, n, typename
21                          std::iterator_traits<IterT>::iterator_category{});
22 }
```

A cela :

```
1  template<ForwardIterator IterT>
2  IterT increment(IterT iter, int n)
3  {
4      assert(n >= 0);
5      for(int i = 0; i < n; ++i)
```

### 3. Conclusion

```
6         ++iter
7
8     return iter;
9 }
10
11 template<RandomAccessIterator IterT>
12 IterT increment(IterT iter, int n)
13 {
14     return iter + n;
15 }
```

La version « Concepts » est clairement plus expressive, on dirait vraiment une simple surcharge sur le type d'itérateurs. Personnellement, voir un code aussi expressif, ça m'émeut énormément !

#### 2.1. Quelques ressources

Je vous conseille de lire les [C++ Core Guidelines](#) concernant l'utilisation de Concepts; cela donne de bons conseils de base pour les utiliser correctement.

A ce propos, je me demande si ça vaut le coup de lire les C++ Core Guidelines en entier. Quelqu'un l'a-t-il déjà fait ? Qu'en pensez-vous ?

J'ai aussi vu une conférence intéressante sur les Concepts il y a quelques semaines, vous pouvez la trouver [ici](#).

---

Pour résumer, les Concepts sont pour moi **la** notion qui manquait pour compléter le système des templates. Ils résolvent de nombreux défauts des templates, et nous donnent de nouveaux outils pour avoir du code plus simple à écrire et à comprendre. C'est clairement, avec les Contrats, ce qui me plaît le plus dans cette nouvelle norme !

### 3. Conclusion

Et voilà pour cet épisode ! Il est un peu plus court que les précédents, j'en suis désolé ; mais je pense que ça va être comme ça pendant encore quelques semaines, car je suis très occupé en ce moment !

A lundi prochain ! <3