C++ TEMPLATE #2

(1/?) Introduction, origin story, possibility & pitfall

(2/?) possibility & pitfall (part 2), Lambda template, template of template, constexpr & design pattern

(3/?) Variadic template? Advanced MetaProg? More?

Reference & const collapsing

- Reference collapsing: https://gcc.godbolt.org/z/5kII8J

  & + & = &

  & + && = &

  && + & = &

  && + && = &&

- Les règles sont conçu pour tout fonctionne correctement et facilement, mais vous risquer de voir des comportement curieux si vous instancier explicitement avec un type.
- Const collapsing: https://gcc.godbolt.org/z/bbi09C
- Support intéressant a retenir: https://www.ibm.com/support/knowledgecenter/en/SSGH3R_16.1.0/com.ibm.xlcpp161.aix.doc/language_ref/reference_collapsing.html

Résolution de surcharge de fonction

- Si une fonction non Template correspond exactement a définition recherché par le compilateur, elle est choisie en priorité : https://gcc.godbolt.org/z/lyzsK-

- Sinon, le compilateur tente de synthétiser tout les templates a sa disposition, et tente de choisir celui qui ‘match’ le mieux:
- Celui dans lequel il y a moins de changement a faire pour coller le type passer en paramètre de fonction et celui passé en paramètre : https://gcc.godbolt.org/z/NJfDX_
- Pars du principe que deux types déclaré séparément dans les paramètre de Template sont sensé être différent: https://gcc.godbolt.org/z/_-iulg

- Si le compilateur n’arrive pas a créer de candidat compilable, ou plusieurs candidats qui semble également valable, une erreur est renvoyée.

Spécialisation

- Permet d'offrir une version particulière du Template pour un set de variable : https://gcc.godbolt.org/z/3Eg1R0
- Les alias ne peuvent pas être spécialisés
- Une spécialisation ne peux exister sans avoir son Template générique déclaré précédemment dans la compilation.
- Une spécialisation est toujours un Template, elle n'introduit pas un nouveau 'symbole' dans le code comme le ferait une surcharge.
- Par conséquent, une spécialisation d'une fonction n'intervient pas quand le compilateur doit choisir entre plusieurs candidats pour une seule expression : https://gcc.godbolt.org/z/fY9Kq8
- **Evitez de faire une spécialisation de fonction !**
- http://www.gotw.ca/publications/mill17.htm

Typename et dependant name

```
template <typename T>
void test(const T& val)
{
    T::name * val;
}
```

- error: missing 'typename' prior to dependent type name ‘T::name'
- Le compilateur doit savoir pour chaque éléments dépendant d’un ‘noms’, si c’est un type ou non, et de base considère que les noms dépendant sont des variable.
- Si le noms dépendant est un type, il doit être précédé par typename.
- Pouvoir garantir qu’une ligne est une déclaration de variable, ou une opération entre deux autres, permet au compilateur de pré-parser les templates pour accélérer la compilation, d’arreter plus rapidement la compilation si il y a un soucis et d’offrir un message d’erreur plus concis.
- **Simplifier vous la vie, utilisez ‘auto’ pour déclaration avec des types complexes**

Lambda template

- Disponible depuis le C++14, c'est une version restreinte des Templates traditionnels.

- Exemple: https://gcc.godbolt.org/z/D_Fc3R

- Il n'est pas vraiment possible de passer explicitement le type
- De fait il n'est pas vraiment possible de faire des paramètre de Template qui soit des valeurs.

Template de template

- Oui, c’est possible, mais la syntaxe n’est pas trivial.
- Exemple: https://gcc.godbolt.org/z/NSoqve
- Les nombre de paramètre Template entre le type passé et le type attendu doivent correspondre, sans tenir compte des paramètre par défaut.

Constexpr

- Fut un temps, les Templates ont été beaucoup utilisé pour calculer des valeurs a la compilation, tout ces usages sont progressivement remplacer par ‘constexpr’

- Permet de calculer des valeurs a la compilation, soit directement en utilisant des variables, ou en passant par une fonction.
- De fait il est possible d’utilisé une expression ‘constexpr’ en paramètre de Template: https://gcc.godbolt.org/z/NW7A_i
- Les fonctions constexpr peuvent être aussi appelé au runtime.

Design pattern avec les Templates

- Design par tag dispatch: https://gcc.godbolt.org/z/qB40MA
- Voir aussi toute le header type_traits

- Adaptors: https://gcc.godbolt.org/z/nUpInc
- Voir aussi, std::priority_queue et std::reverse_iterator

- Policy Based design, ex : allocator

Ressource pour aller plus loin

- **CppCon 2018: Walter E. Brown “C++ Function Templates: How Do They Really Work?”** https://www.youtube.com/watch?v=NIDEjY5ywqU
- **CppCon 2016: Arthur O'Dwyer “Template Normal Programming (part 1 of 2)”** **https://www.youtube.com/watch?v=vwrXHznaYLA**
- **CppCon 2016: Arthur O'Dwyer “Template Normal Programming (part 2 of 2)"** **https://www.youtube.com/watch?v=VIz6xBvwYd8**

Plus de question ?