



Петър Армянов

Доцент във ФМИ на СУ

Концепциите в C++20

Как да направим шаблоните по-приветливи

Партньори:



Следете актуалните обяви за **C, C++**

DEV.BG



Малко мотивация

- Какво си мислим, когато чуем за шаблони (templates) в C++?
 - и особено ако става дума за метапрограмиране...



Малко мотивация

- Какво си мислим, когато чуем за шаблони (templates) в C++?
 - и особено ако става дума за метапрограмиране...





Малко мотивация

- Какво си мислим, когато чуем за шаблони (templates) в C++?
 - и особено ако става дума за метапрограмиране...
- А защо е така?



Малко мотивация

- Какво си мислим, когато чуем за шаблони (templates) в C++?
 - и особено ако става дума за метапрограмиране...
- А защо е така?

Прост пример за функция print



Изискване, ограничение, концепция

- Какво е изискване (requirement)?
 - изчислим по вре. на компилация булев израз
- Какво е ограничение (constraint)?
 - използване на изискване при инстанциране
- Какво е концепция (concept)?
 - именовано ограничение



Първо ограничение

```
void print(const T& a) {  
    std::cout << a << "\n";  
}
```

А ако T е указател:

```
std::cout << *a << "\n";
```



Още малко с това ограничение...

Max (A, B);

Партньори:



BOSCH
Invented for life



Следете актуалните обяви за **C, C++**

DEV.BG



По-сложни ограничения

```
auto Max(IsPointer auto a, IsPointer auto b)
    requires std::totally_ordered_with<decltype(*a), decltype(*b)>
{
    return Max(*a, *b);
}
```



Концепции с по-сложни ограничения

```
template<typename T>
concept IsPointer = requires(T p) {
    *p;
    p == nullptr;
    {p < p} -> std::same_as<bool>;
};
```



Израз на изискване

директни - булев израз:

```
template<typename T>  
requires (sizeof(int) != sizeof(long))
```

```
template<typename T>  
requires  
std::integral<std::remove_reference_t<decltype(*std::declval<T>())>>
```



Израз на изискване (2)

прости:

```
template<typename T1, typename T2>
concept ValuePointer =
    requires(T1 val, T2 p) {
        *p;           // operator*
        p[0];         // operator[] (int)
        p->value();    // member function value()
        *p > val;      // comparison of operator* with T1
        p == nullptr; // comparison of a T2 with a nullptr
    };
```



Израз на изискване (3)

За типове - дали даден тип може да бъде създаден:

```
... requires {  
    typename MyType1<T1>;  
    typename T1::value_type;  
    typename std::ranges::iterator_t<T1>;  
    typename std::common_type_t<T1, T2>;  
}
```



Израз на изискване (4)

Съставни изисквания:

```
... requires (T x) {  
    { &x } -> std::input_or_output_iterator;  
    { x == x };  
    { x == x } -> std::convertible_to<bool>;  
    { x == x }noexcept;  
    { x == x }noexcept -> std::convertible_to<bool>;  
}
```

Вложени изисквания



Клауза или израз?

```
template<typename T>
    requires requires(T p) {
        *p;
        p == nullptr;
        {p < p} -> std::same_as<bool>;
    }
void f(const T&) {...}
```



Малко повече синтаксис (1)

- Можем да налагаме ограничения към:
 - Функции
 - Класове
 - Кратки имена (алиаси)
 - Шаблонни константи
 - Член функции (но шаблонни!)



Малко повече синтаксис (2)

- Нетипови параметри (Value parameters)
 - При директно посочване ограничават само типа, но не и стойността!
- Могат да се използват навсякъде, където трябва `bool`
 - Всякакви изрази, но особено `if constexpr`



Концепции и auto

- `void foo(const std::integral auto& val) { ... }`
- `std::floating_point auto val1 = f();`
- `for (const std::integral auto& elem : coll) { ... }`
- `std::copyable auto foo(auto) { ... }`
- `template<typename T, std::integral auto Max>`
`class MyClass{...};`



Уточняване (subsume)

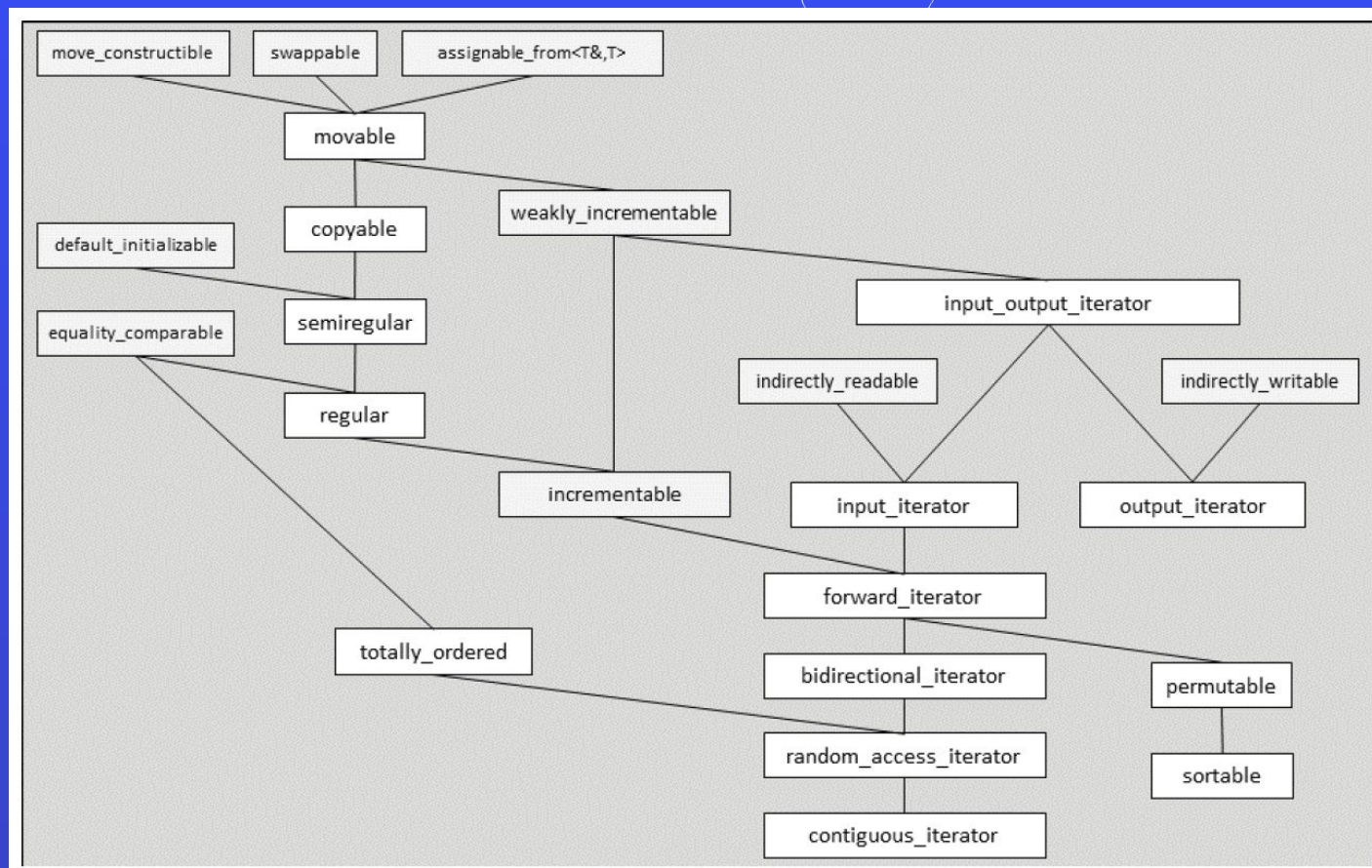
- Много полезна практика - компилаторът разпознава най-ясната възможна версия и избира нея.

Пример:

- Неявно уточняване
- Внимавайте с комутативните концепции!



Граф на стандартните концепции



ref: Nicolai M. Josuttis,
C++20 - The Complete Guide



Препоръки (1)

- Групирайте ограниченията в именовани концепции
- Внимавайте с комутативните концепции
- Използвайте максимално стандартните концепции
- По-добре концепции пред булеви изрази и traits, защото
 - могат да се уточняват
 - могат да се използват и като типови ограничения
 - могат да се използват в `if constexpr`
- Не разчитайте прекалено на уточняване, по-добре пишете по-прецизни ограничения



Препоръки (2)

- Избягвайте използването на || при изброяване на клаузи за изисквания
- Избягвайте клаузи, които са винаги верни:

```
template<typename T>
```

```
... requires { std::is_const_v<T>; }
```



Типични грешки (1)

- Изразите се проверяват за валидност, а не за вярност!
- || не прави каквото очакваме в израз за изискване!

```
template<typename T1, typename T2>
    ... requires(T1 val, T2 p) {
        *p > val || p == nullptr
    }
```

трябва да е:

```
template<typename T1, typename T2>
    ... requires(T1 val, T2 p) { *p > val; } ||
        requires(T2 p) { p == nullptr; }
};
```



Типични грешки (2)

- Типовите изисквания не проверяват за налични имплементации, само за валидност!

```
template<typename T>
    concept StdHash = requires {
        typename std::hash<T>;
    };
```

```
template<typename T>
    concept StdHash = requires {
        std::hash<T> {};
    };
```




ИЗТОЧНИЦИ

- https://github.com/peshe/_CPP20_Concepts
- Nicolai M. Josuttis, C++20 - The Complete Guide
- https://www.youtube.com/watch?v=_doRiQS4GS8
- <https://www.youtube.com/watch?v=qawSiMIXtE4>



Благодаря за вниманието!

Въпроси?

e-mail: armianov@gmail.com

github: <https://github.com/peshe/>

Партньори:



Следете актуалните обяви за **C, C++**

DEV.BG







Thank you!

Contacts:

 <https://www.linkedin.com/in/peshe/>

 <https://github.com/peshe/>

СЛЕДВАЩО СЪБИТИЕ

					
Лектор	Дата	Език			

Партньори:



BOSCH
Invented for life



Следете актуалните обяви за **C, C++**

DEV.BG