OTUS

T.120: Use template metaprogramming only when you really need to

Дмитрий Шебордаев, Ведущий специалист

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Дмитрий Шебордаев, Ведущий специалист

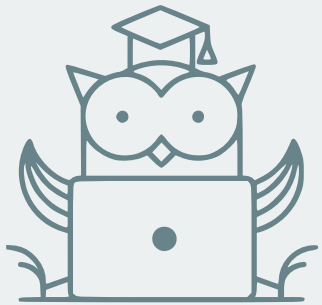
OTUS

Введение

Template MetaProgramming (TMP)

Дмитрий Шебордаев

Ведущий специалист



Сегодня

- Эволюция TMP
- Что имеем сейчас
- Читаем STL

Чем раньше, тем лучше

```
for(auto i: v)
{
    std::string tmp;
    // ...
};

std::string tmp;
for(auto i: v)
{
    // ...
};
```


Чем раньше, тем лучше

```
void thread()
{
    // 1 ...
    while(true)
    {
        auto message = receive();
        // 2 ...
    };
};
```


Чем раньше, тем лучше

```
int main()
{
    std::vector<std::string> v; // std::list<std::string>
    for (std::string line; std::getline(std::in, line); )
    {
        v.push_back(s);
    }
};
```


Чем раньше, тем лучше

```
int main()
{
    std::vector<std::string> v;
    size_t r;
    std::cin >> r;
    v.reserve(r);
    for (std::string line; std::getline(std::in, line); )
    {
        v.push_back(s);
    }
};
```


Что насчет времени компиляции

- Препроцессор
- Генерация исходников
- Шаблоны
- constexpr

Что насчет времени компиляции

```
int main()
{
    size_t seconds = 24 * 60 * 60;
    size_t seconds_high_perfomance = 86400; // 24 * 60 * 60
};
```


template vs class

```
struct foo
{
    static const int value = 1000;
};

template<int P>
struct foo
{
    static const int value = 2000;
};

foo::value; // ужесуществует
foo<10>::value; // толькочтородился
```


metafunction call

```
template<int P>
struct foo
{
    static const int value = P;
};

foo<10>::value

/* constexpr */ int foo(int P)
{
    return P;
};

foo(10)
```


expr

```
template<int P>
struct abs
{
    static const int value = P < 0 ? -P : P;
};

abs<-10>::value
abs<10>::value
```


recursion

```
template<int P>
struct fact
{
    // static_assert(P > 0, "positive expected");
    static const int value = P * fact<P - 1>::value;
};

template<>
struct fact<0>
{
    static const int value = 1;
};

fact<11>::value;
fact<-1>::value; // ???
```


type as param

```
template<typename T>
struct is_int
{
    static const bool value = false;
};

template<>
struct is_int<int>
{
    static const bool value = true;
};
```


type as result

```
template<typename T>
struct remove_const
{
    using type = T;
};

template<typename T>
struct remove_const<const T>
{
    using type = T;
};

remove_const<int>::type a1;
remove_const<const int>::type a2;
```


Соглашения

- is_integral_v<T>
- is_integral<T>::value
- remove_const_t<int>
- remove_const<int>::type

Наследование

```
template<typename T>
struct type_is
{
    using type = T;
};

template<typename T>
struct remove_const : type_is<T> {};

template<typename T>
struct remove_const<const T> : type_is<T> {};
```


Ветвление

```
template<bool B, class T, class F>
struct conditional : type_is<T> {};

template<class T, class F>
struct conditional<false, T, F> : type_is<F> {};

conditional_t<cfg_use_fast, long, double> total;
conditional_t<cfg_use_fast, less_prec, more_prec>{}(total);
class foo : public conditional_t<cfg_use_fast, bar, baz>
```


SFINAE

```
template<bool B, class T>
struct enable_if : type_is<T> {};

template<class T>
struct enable_if<false, T> {};

enable_if<false, int>::type
```


SFINAE

```
template<class T>
enable_if_t<is_integral_v<T>, std::string> foo(const T &);

template<class T>
enable_if_t<is_floating_point_v<T>, std::string> foo(const T &

foo(1);
foo(1.0);
foo("1");
```


Опять наследование

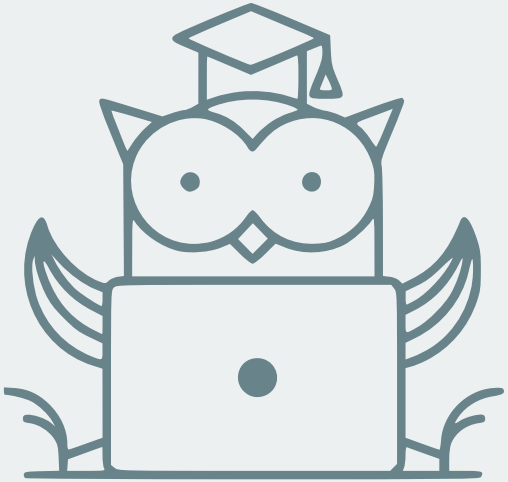
```
template<typename T, T v>
struct integral_constant {
    static constexpr T value = v;
    using type = integral_constant;
};

template <bool B>
using bool_constant = integral_constant<bool, B>;

using true_type = bool_constant<true>;
using false_type = bool_constant<false>;
```


Цикл

```
template<typename T, typename...Args>
struct is_one_of;

template<typename T>
struct is_one_of<T> : false_type {};

template<typename T, typename...Args>
struct is_one_of<T, T, Args...> : true_type {};

template <typename T, typename U, typename...Args>
struct is_one_of<T, U, Args...> : is_one_of<T, Args...> {};
```


почитать

- http://en.cppreference.com/w/cpp/header/type_traits



Спасибо за внимание!