Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

Template metaprogramming (TMP)

Андрей Рыжиков

10 лет опыта разработки на C++ (обработка изображений)

https://t.me/ryzhikovas

План вебинара

- Эволюция MP в C++
- Аллокатор
- Кирпичики для TMP в STL
- SFINAE
- Современное положение дел

Цели вебинара

К концу занятия вы

1. Будете понимать, что такое метапрограммирование в C++
2. Сможете без страха читать и писать шаблонные конструкции, связанные с TMP
3. Ознакомитесь со “строительными блоками” TMP из STL
4. Уясните механизм SFINAE

Аллокатор

```
 1 template<typename T>
 2 struct Allocator {
 3     T *allocate(std::size_t n);
 4
 5     void deallocate(T *p, std::size_t n);
 6
 7     template<typename U, typename ...Args>
 8     void construct(U *p, Args &&...args);
 9
10     template<typename U>
11     void destroy(U *p);
12 };
```

reserve?

`std::vector<...>, std::array<...>,`
`c-style array`

Q: Что у них общего?

A: Линейно расположены в памяти.

- - Доступ по индексу за O(1)
- - Cache-friendly

reserve?

`std::list<...>, std::set<...>, std::map<...>`

Данные расположены нелинейно.

- Медленный произвольный доступ
- Ломают кэш

C++ Core Guidelines

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

Код, который создает код

Метапрограммирование - создание программ, которые порождают другие программы как результат своей работы.

Что можем в compile-time?

- Внешний кодогенератор
- Препроцессор
- Шаблоны
- constexpr

Q: Что такое **DRY**?

A: Don’t repeat yourself, не повторяйся

Шаблон как кодогенератор

```
 1 struct foo {
 2     static const int a = 10;
 3 };
 4
 5 template<int v>
 6 struct off {
 7     static const int b = 10;
 8 };
 9
10 foo::a;    // есть всегда
11 off<1>::b; // появляется здесь
12 off<2>::b;
```

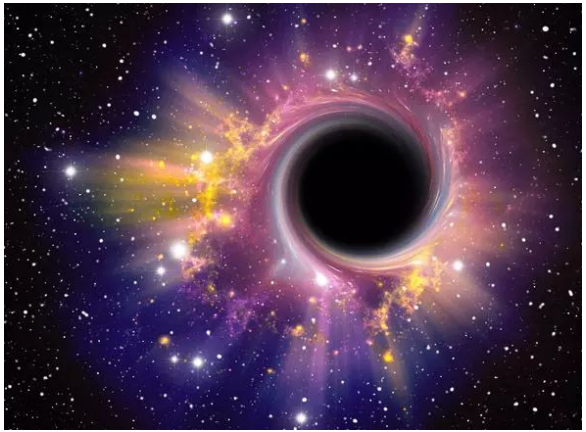
Q: `&off<1>::b == &off<2>::b`?

Шаблон как кодогенератор. Fix it

```
 1 #include <iostream>
 2
 3 template<int v>
 4 struct off {
 5     static const int b = 10;
 6 };
 7
 8 int main() {
 9    std::cout << std::boolalpha
10         << (off<1>::b == off<2>::b) << std::endl
11         << (&off<1>::b == &off<2>::b) << std::endl;
12 }
```

Constexpr-функция

```
 1 constexpr int foo(int v) {
 2     return v * v;
 3 }
 4
 5 int main(int argc, char** argv) {
 6     // runtime
 7     std::cout << foo(argc) << endl;
 8     // compile-time
 9     std::array<int, foo(5)> array;
10 }
```



Метафункции

```
1  constexpr int foo(int v) {
2      return v;
3  }
4
5  template<int v>
6  struct foo {
7      static const int value = v;
8  };
9
10 foo<10>::value; // compile-time
```

Условия

```
 1 template<int v>
 2 struct abs {
 3     static const int value = v < 0 ? -v : v;
 4 };
 5
 6 abs<-10>::value;
 7 abs<10>::value;
 8
 9 static const int value = [] () {
10     if (v < 0) {
11         return -v;
12     }
13     return v;
14 }();
```

Циклы

```
 1 template<int p>
 2 struct fact {
 3     static const int v = p * fact<p-1>::v;
 4 };
 5
 6 template<>
 7 struct fact<0> {
 8    static const int v = 1;
 9 };
10
11 fact<11>::v;
12 fact<-2>::v;
```

Тип как параметр

```
 1 template<typename T>
 2 struct is_int {
 3     static const bool value = false;
 4 };
 5
 6 template<>
 7 struct is_int<int> {
 8     static const bool value = true;
 9 };
10
11 is_int<int>::value;
```

Тип как результат

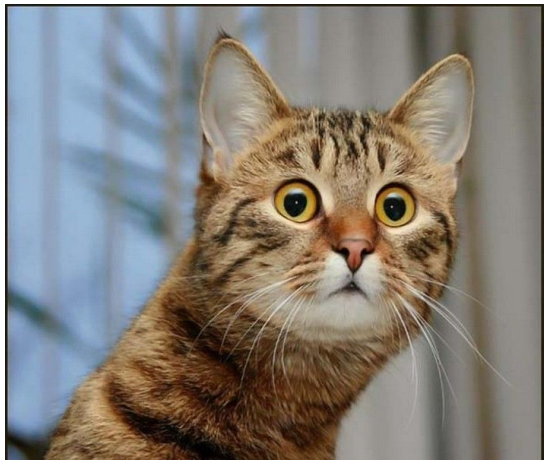
```
 1 template<typename T>
 2 struct remove_const {
 3     using type = T;
 4 };
 5
 6 template<typename U>
 7 struct remove_const<const U> {
 8    using type = U;
 9 };
10
11 remove_const<int>::type a1;
12 remove_const<const int>::type a2;
```

Соглашения по именованию в STL

```
1 std::is_integral<T>::value
2 std::remove_const<T>::type
3 std::is_integral_v<T>
4 std::remove_const_t<T>
```

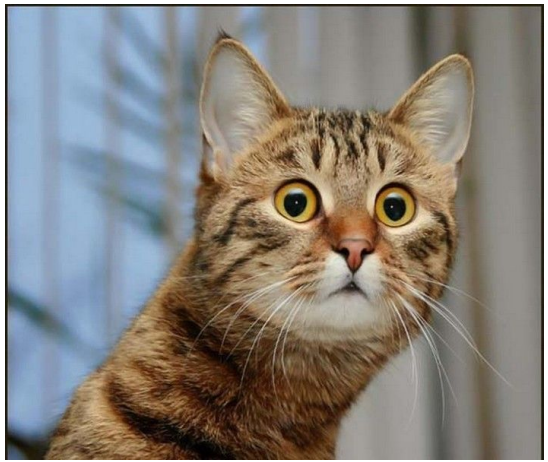
Наследование

```
 1 template<typename T>
 2 struct type_is {
 3     using type = T;
 4 };
 5
 6 template<typename T>
 7 struct remove_const: type_is<T> {};
 8
 9 template<typename T>
10 struct remove_const<const T>: type_is<T> {};
11
12 template<typename T>
13 using remove_const_t = typename remove_const<T>::type;
```



Ветвление

```
1 template<bool cond, class T, class F>
2 struct conditional:
      type_is<T> {};
3
4 template<class T, class F>
5 struct conditional<false, T, F>:
      type_is<F> {};
```



SFINAE

```
1 template<bool cond, class T>
2 struct enable_if:
      type_is<T> {};
3
4 template<class T>
5 struct enable_if<false, T>{};
6
7 enable_if<false, int>::type;
```

SFINAE: substitution failure is not an error.


БЛИЖЕ К ЖИЗНИ,
К ЖИВОМУ ДЕЛУ!

Вопросы?

Слушаю/читаю в чате

Цели вебинара

К концу занятия вы

1. Будете понимать, что такое метапрограммирование в C++
2. Сможете без страха читать и писать шаблонные конструкции, связанные с TMP
3. Ознакомитесь со “строительными блоками” TMP из STL
4. Уясните механизм SFINAE

Заполните, пожалуйста, опрос о занятии по ссылке в чате