Яндекс Такси

Итоги встречи в Праге

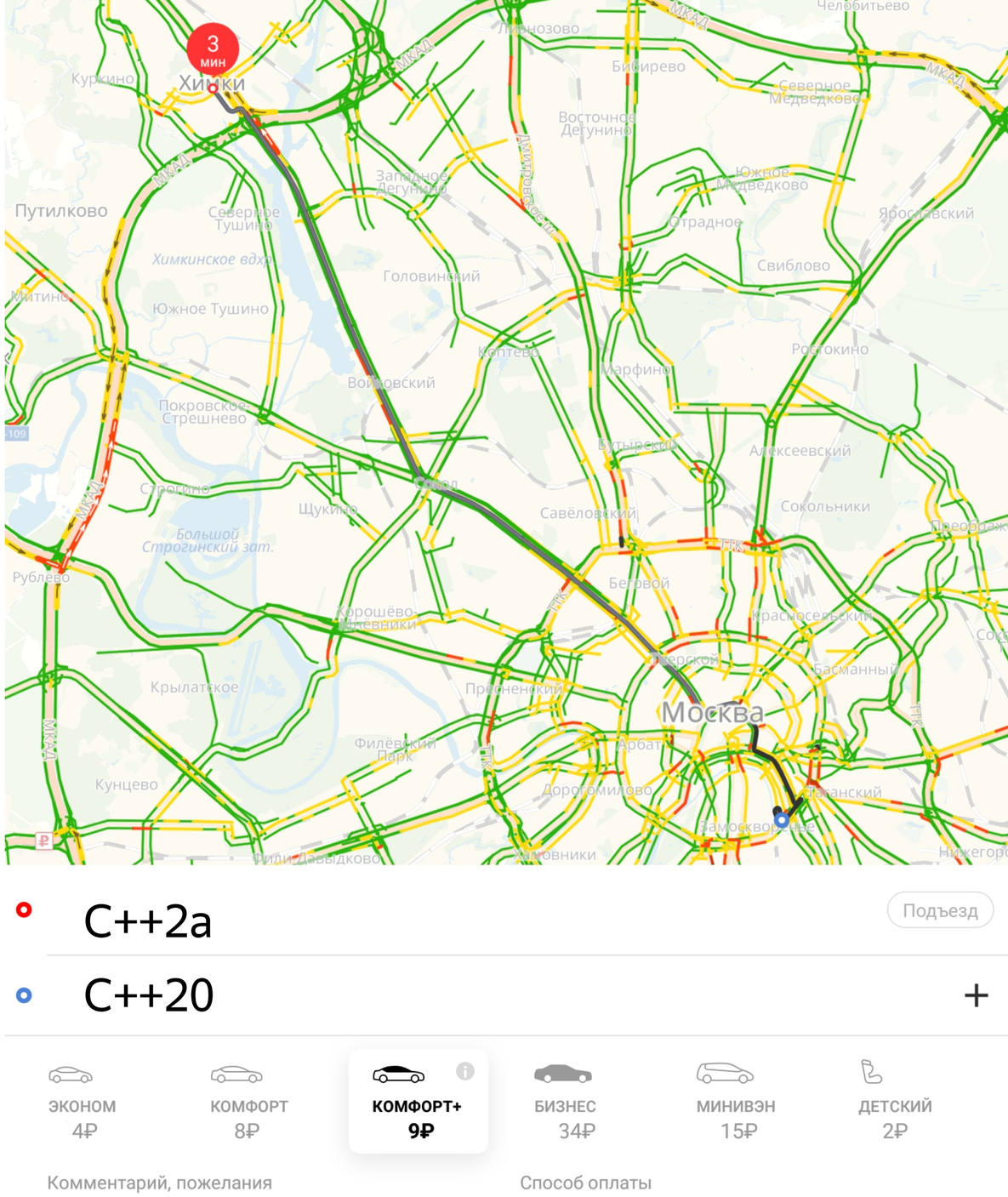
Полухин Антон

Antony Polukhin

Яндекс Такси

Содержание

- Мелочи C и bool
- Крупные новинки C++20
  - std::format
  - Concepts
  - Календарь
  - Modules
  - Coroutines
  - Ranges
- Планы на карантине



Мелочи C и bool

Меньше UB (1)

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();
```

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();


struct data_type {
  int i;
  short s;
};
```

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();


struct data_type {
  int i;
  short s;
};


void example1() {




}
```

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();


struct data_type {
  int i;
  short s;
};


void example1() {
  auto raw_data = get_from_network();



}
```

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();


struct data_type {
  int i;
  short s;
};


void example1() {
  auto raw_data = get_from_network();
  auto& data = *reinterpret_cast<data_type*>(raw_data.get());


}
```

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();


struct data_type {
  int i;
  short s;
};


void example1() {
  auto raw_data = get_from_network();
  auto& data = *reinterpret_cast<data_type*>(raw_data.get());
  assert(data.i == 42);
}
```

Меньше UB (1)

```
std::unique_ptr<std::byte[]> get_from_network();


struct data_type {
  int i;
  short s;
};


void example1() {
  auto raw_data = get_from_network();
  auto& data = *reinterpret_cast<data_type*>(raw_data.get());
  assert(data.i == 42);  // This is fine!
}
```

Меньше UB (2)

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
```

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
std::size_t read_network_data(std::span<unsigned char> out_buf);
```

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
std::size_t read_network_data(std::span<unsigned char> out_buf);


void example2() {




}
```

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
std::size_t read_network_data(std::span<unsigned char> out_buf);


void example2() {
  unsigned char buffer[256];




}
```

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
std::size_t read_network_data(std::span<unsigned char> out_buf);


void example2() {
  unsigned char buffer[256];
  const auto size = read_network_data(buffer);



}
```

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
std::size_t read_network_data(std::span<unsigned char> out_buf);


void example2() {
  unsigned char buffer[256];
  const auto size = read_network_data(buffer);
  auto& data = *reinterpret_cast<data_type*>(&buffer[0]);
  assert(data.i == 42);
}
```

Меньше UB (2)

```
struct data_type {
  int i;
  short s;
};
std::size_t read_network_data(std::span<unsigned char> out_buf);


void example2() {
  unsigned char buffer[256];
  const auto size = read_network_data(buffer);
  auto& data = *reinterpret_cast<data_type*>(&buffer[0]);
  assert(data.i == 42);  // This is fine!
}
```

Меньше проблем

```
bool example3(int* i) {
  bool b{i};
  return b;
}
```

Меньше проблем

```
bool example3(int* i) {
  bool b{i};  // ???
  return b;
}
```

Меньше проблем

```
bool example3(int* i) {
  bool b{i};  // C++20 → ill formed
  return b;
}
```

Меньше проблем

```
bool example3(bool* begin, bool* end) {
  return std::set<bool>{begin, end}.size() == 1;
}
```

Крупные новинки C++20

Итоги встречи в Праге

std::format

Format Inro

Format Inro

```
std::string res0 = std::format("{} from {}", "Hello", "Russia");
```

Format Inro

```
std::string res0 = std::format("{} from {}", "Hello", "Russia");


std::string res1 = std::format("{1} from {0}", "Russia", "Hello");
```

Format Inro

```
std::string res0 = std::format("{} from {}", "Hello", "Russia");


std::string res1 = std::format("{1} from {0}", "Russia", "Hello");


int width = 10;
int precision = 3;
std::string s = std::format("{0:{1}.{2}f}", 12.345678, width, precision); // "    12.346"
```

Format Inro

```
std::string res0 = std::format("{} from {}", "Hello", "Russia");


std::string res1 = std::format("{1} from {0}", "Russia", "Hello");


int width = 10;
int precision = 3;
std::string s = std::format("{0:{1}.{2}f}", 12.345678, width, precision); // "    12.346"


std::array<char, 200> buffer;
std::format_to_n(buffer.data(), buffer().size(), "{0:b} {0:d} {0:o} {0:x}", 42);
assert(buffer.data() == "101010 42 52 2a"sv);
```

Format Inro

```
int width = 10;

int precision = 3;

std::cout << std::format("{0:{1}.{2}f}", 12.345678, width, precision); // "    12.346"
```

Format Inro

```
int width = 10;

int precision = 3;

std::cout << std::format("{0:{1}.{2}f}", 12.345678, width, precision); // "    12.346"
```

Концепты!

Концепты

Концепты

```
template <class Container>
void reserve(Container& container, std::size_t size) {




}
```

Концепты

```
template <class Container>
void reserve(Container& container, std::size_t size) {
  if constexpr (                                    ) {
    container.reserve(size);
  }
}
```

Концепты

```
template <class Container>
void reserve(Container& container, std::size_t size) {
  if constexpr (requires {container.reserve(size);}) {
    container.reserve(size);
  }
}
```

Концепты

```
#include <array>
#include <vector>


template <class Container>
void reserve(Container& container, std::size_t size)


auto example4() {
    std::array<char, 512> vec;
    reserve(vec, 100);
    std::vector<int> arr;
    reserve(arr, 100);
}
```

Календарь!

Календарь

```
#include <chrono>
```

Календарь

```
#include <chrono>

auto example5() {
  using namespace std::chrono;
  using namespace std::chrono_literals;




}
```

Календарь

```
#include <chrono>

auto example5() {
  using namespace std::chrono;
  using namespace std::chrono_literals;


  // 2020-05-29




}
```

Календарь

```
#include <chrono>

auto example5() {
  using namespace std::chrono;
  using namespace std::chrono_literals;


  // 2020-05-29
  year_month_day ymd = 2020y / May / 29d;




}
```

Календарь

```
#include <chrono>

auto example5() {
  using namespace std::chrono;
  using namespace std::chrono_literals;


  // 2020-05-29
  year_month_day ymd = 2020y / May / 29d;


  // 2020-05-29 07:30:06.153 UTC



}
```

Календарь

```
#include <chrono>

auto example5() {
  using namespace std::chrono;
  using namespace std::chrono_literals;


  // 2020-05-29
  year_month_day ymd = 2020y / May / 29d;


  // 2020-05-29 07:30:06.153 UTC
  auto tp = sys_days{ymd} + 7h + 30min + 6s + 153ms;


}
```

Календарь

```
#include <chrono>
auto example5() {
  using namespace std::chrono;
  using namespace std::chrono_literals;

  // 2020-05-29
  year_month_day ymd = 2020y / May / 29d;

  // 2020-05-29 07:30:06.153 UTC
  auto tp = sys_days{ymd} + 7h + 30min + 6s + 153ms;
  std::cout << zoned_time{"Asia/Tokyo", tp} << '\n';  // 2020-05-29 16:30:06.153 JST
}
```

Modules!

Модули

Модули

```
namespace userver::detail { /*...*/ }

namespace utils::impl { /*...*/ }
```

Корутины!

Coroutines Inro

```
awaitable<void> echo(tcp::socket socket) {
  try {
    char data[1024];
    for (;;) {
      std::size_t n = co_await socket.async_read_some(boost::asio::buffer(data),
                                                      use_awaitable);
      co_await async_write(socket, boost::asio::buffer(data, n), use_awaitable);
    }
  } catch (std::exception& e) {
    std::printf("echo Exception: %s\n", e.what());
  }
}
```

Coroutines Inro

```
awaitable<void> echo(tcp::socket socket) {
  try {
    char data[1024];
    for (;;) {
      std::size_t n = co_await socket.async_read_some(boost::asio::buffer(data),
                                                      use_awaitable);
      co_await async_write(socket, boost::asio::buffer(data, n), use_awaitable);
    }
  } catch (std::exception& e) {
    std::printf("echo Exception: %s\n", e.what());
  }
}
```

Coroutines links

- ASIO/Boost.ASIO
- CppCoro https://github.com/lewissbaker/cppcoro

Ranges!

Ranges

```
std::ranges::sort(container);
```

Ranges

```
#include <ranges>


template <class Range>
auto Eval(Range& r);  // return container


template <class T>
auto example7(const T& data) {
  using std::ranges::views::join;
  using std::ranges::views::transform;


  return Eval(  /*...*/ );
}
```

Ranges

```
return Eval(  //
    data      //
    | transform([](const auto& grid) {
        return Eval(grid | transform([&grid](const auto& val) {
                       return std::make_tuple(&grid, val);
                   }));
      })    //
    | join  //
    | transform([](const auto& val) {
      return Eval(val | transform([val](const auto& v) { return /*...*/; }));
    })      //
    | join  //
```

Планы на карантине

WG21 Plans

WG21 Plans

- ???

WG21 Plans

- ???
- ???

WG21 Plans

- ???
- ???
- ???

Есть идеи C++23?

https://stdcpp.ru

C++
РГ21 C++ РОССИЯ

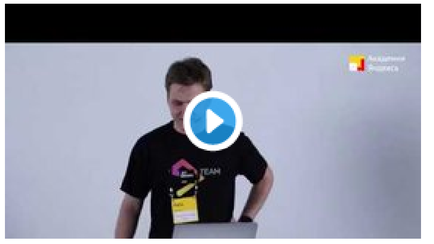
RSS Ru En yndx-antoshkka, выход

- О проекте
- Новости
- Предложения
- Вопросы и ответы
- Инструкция по подготовке proposal

stdcppru
@stdcppru

Для тех, кто пропустил встречу в декабре:
– Обзор встречи Комитета по стандартизации C++ в Сан-Диего, США – Антон Полухин
youtube.com/watch?v=QaDO9L…
– Модули в C++ – Дмитрий Кожевников
youtube.com/watch?v=p8MkTJ…

YouTube @YouTube



9 янв. 2019 г.

Помогаем готовить предложения для защиты перед рабочей группой Комитета ISO C++

Новости

27 ноября 2018 29 мая 2018 14 марта 2018

Спасибо

Полухин Антон

Эксперт-разработчик C++

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/

РГ21 C++ РОССИЯ

Полухин Антон

Эксперт-разработчик C++

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/

РГ21 C++ РОССИЯ

https://t.me/CppQuizzBot

Quizz

Вопрос 1 из 8:

Как зовут создателя языка C++?

Правильный ответ: Бьёрн Страуструп 19:33

Вопрос 2 из 8:

Какой стандарт C++ ввёл move-семантику

Правильный ответ: C++11 19:33

Вопрос 3 из 8:

Как называется `T&&` в `template <class T> void foo(T&&)`

Правильный ответ: forwarding/universal reference 19:33

Вопрос 4 из 8:

Для чего может быть нужна конструкция:

```
/// Example: template <class T> void foo(rvalue_t<T> val);
///
template <class T, class = std::enable_if_t<
    std::is_rvalue_reference_v<T&&>
>>
using rvalue_t = T&&;
```

Правильный ответ: чтобы принимать только rvalue 19:33

Вопрос 5 из 8:

Вы пишите свой mutex. Какие оптимальные memory order должны быть на операциях lock/unlock, чтобы пользовательский код продолжал работать корректно.

Правильный ответ: acquire/release 19:33

Вопрос 6 из 8:

Для многопоточной работы с `std::atomic<int> i{0};` есть ли разница в вызовах `i.load(relaxed)` и `i.fetch_add(0, relaxed)`?

Правильный ответ: fetch_add - это rmv-операция, результат будет актуальным 19:33

Вопрос 7 из 8:

Какой тип получится у `x` в результате выполнения:

```
int i;
auto x = std::make_tuple(std::ref(i), std::cref(i));
```

Правильный ответ: std::tuple<int&, const int&> 19:33

Вопрос 8 из 8:

Какой тип получится у `x` в C++17 в результате выполнения:

```
int i;
auto x = std::tuple(std::ref(i), std::cref(i));
```

Правильный ответ: tuple<reference_wrapper<int>, reference_wrapper<const int>> 19:33