C++ Zero Cost Conf */

Новости РГ 21

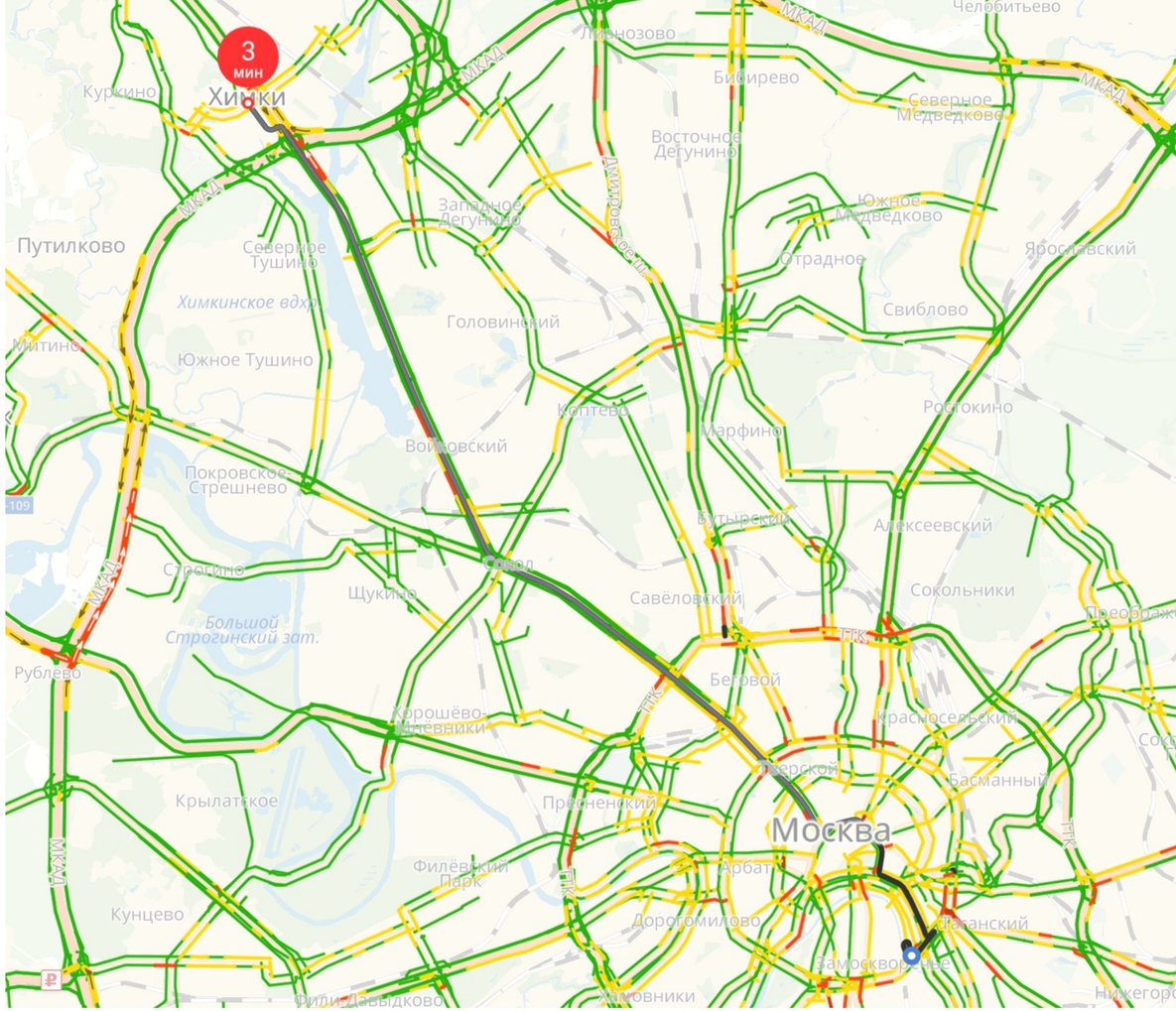
Летняя встреча ISO WG21

Полухин Антон
Эксперт разработчик C++

Yandex for developers *//>

Содержание

- #warning
- volatile
- constexpr
- static operator()
- float16_t и bfloat16_t
- [[assume(x)]]
- start_lifetime_as
- forward_like
- print
- flat_set / flat_map
- mdspan



Совместимость с C

Совместимость с C

```
#warning Prefer including A.hpp or B.hpp instead of AB.hpp
```

Совместимость с C

```
#warning Prefer including A.hpp or B.hpp instead of AB.hpp

// На подходе в C++26:
constexpr std::byte image[] = {
   #embed "avatar.png"
};
```

Совместимость с C

```
#warning Prefer including A.hpp or B.hpp instead of AB.hpp

// На подходе в C++26:
constexpr std::byte image[] = {
   #embed "avatar.png"
};

// На подходе в C++26:
std::int128_t
std::int256_t
std::int512_t
```

volatile

volatile

```
UART1->UCSR0B |= (1<<UCSZ01 );
UART1->UCSR0B &= (1<<UCSZ02 );
UART1->UCSR0B ^= (1<<UCSZ03 );
```

constexpr to_chars/from_chars

static operator()

std::float16_t, std::bfloat16_t
std::float128_t

```
[[assume(!std::isinf(x))]]
```

start_lifetime_as

start_lifetime_as

```
struct ProtocolHeader {
  unsigned char version;
  unsigned char msg_type;
  unsigned char chunks_count;
};
```

start_lifetime_as

```cpp
struct ProtocolHeader {
  unsigned char version;
  unsigned char msg_type;
  unsigned char chunks_count;
};

void ReceiveData(std::span<std::byte> data_from_net) {
    if (data_from_net.size() < sizeof(ProtocolHeader)) throw SomeException();
    const auto* header = std::start_lifetime_as<ProtocolHeader>(
        data_from_net.data()
    );
    switch (header->msg_type) {
        // ...
    }
}
```

start_lifetime_as

```
struct ProtocolHeader {
  unsigned char version;
  unsigned char msg_type;
  unsigned char chunks_count;
};

void ReceiveData(std::span<std::byte> data_from_net) {
    if (data_from_net.size() < sizeof(ProtocolHeader)) throw SomeException();
    const auto* header = std::start_lifetime_as<ProtocolHeader>(
        data_from_net.data()
    );
    switch (header->msg_type) {
        // ...
    }
}
```

forward_like

forward_like

```
class Something {
 public:
  auto operator*(this auto&& self) {
    return std::forward_like<decltype(self)>(data_);
  }
 private:
  std::string data_;
};
```

forward_like

```
class Something {
 public:
  auto operator*(this auto&& self) {
    return std::forward_like<decltype(self)>(data_);
  }
 private:
  std::string data_;
};
```

print

print

```
std::print("Привет, {}! У вас {} писем", username, email_count);
```

flat_map / flat_set

mdspan

Спасибо

Полухин Антон

Эксперт-разработчик C++

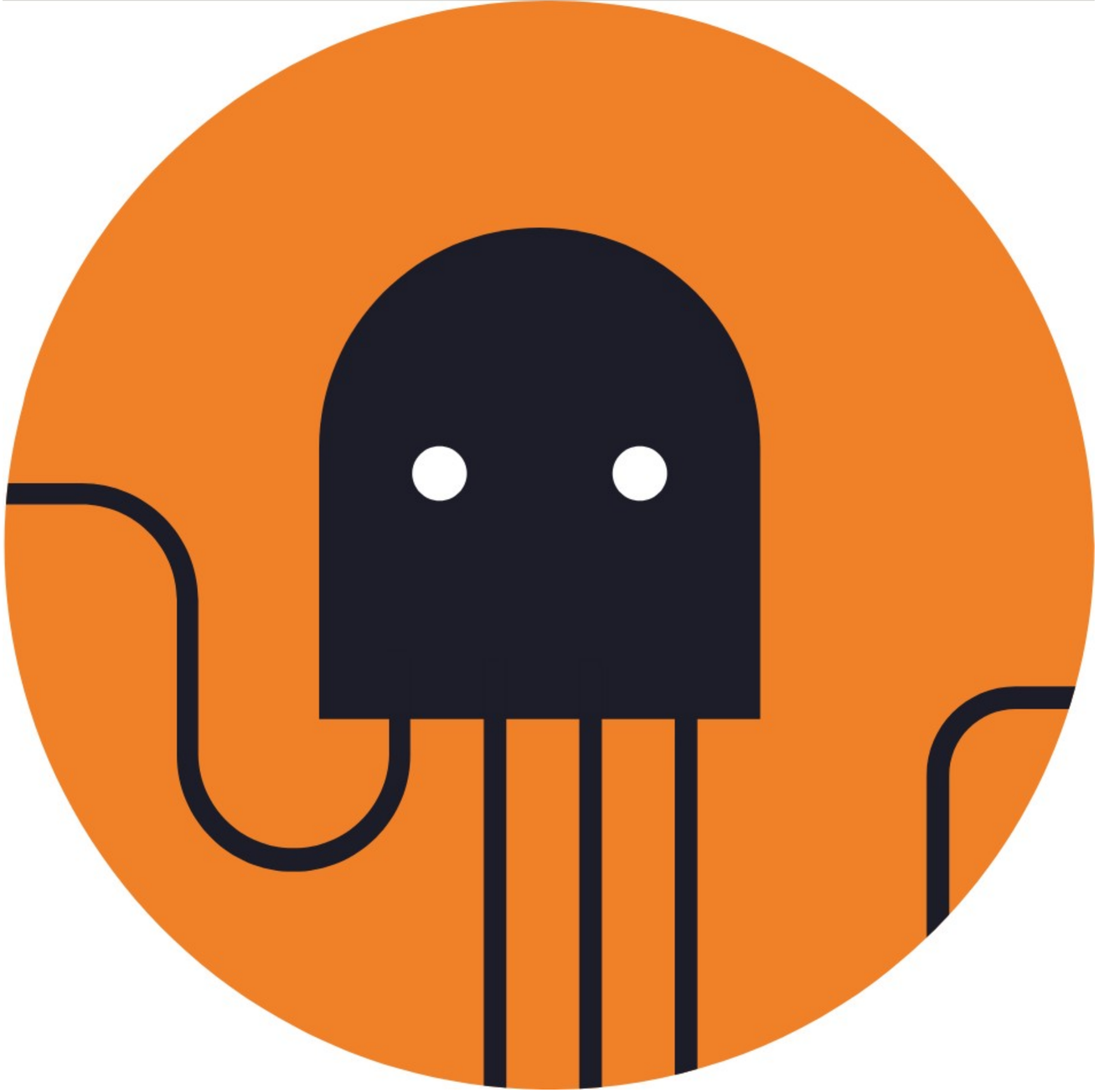
antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/

РГ21 C++ РОССИЯ



https://github.com/userver-framework