Ключевые фичи C++20

+ немного про C++17 и C++23

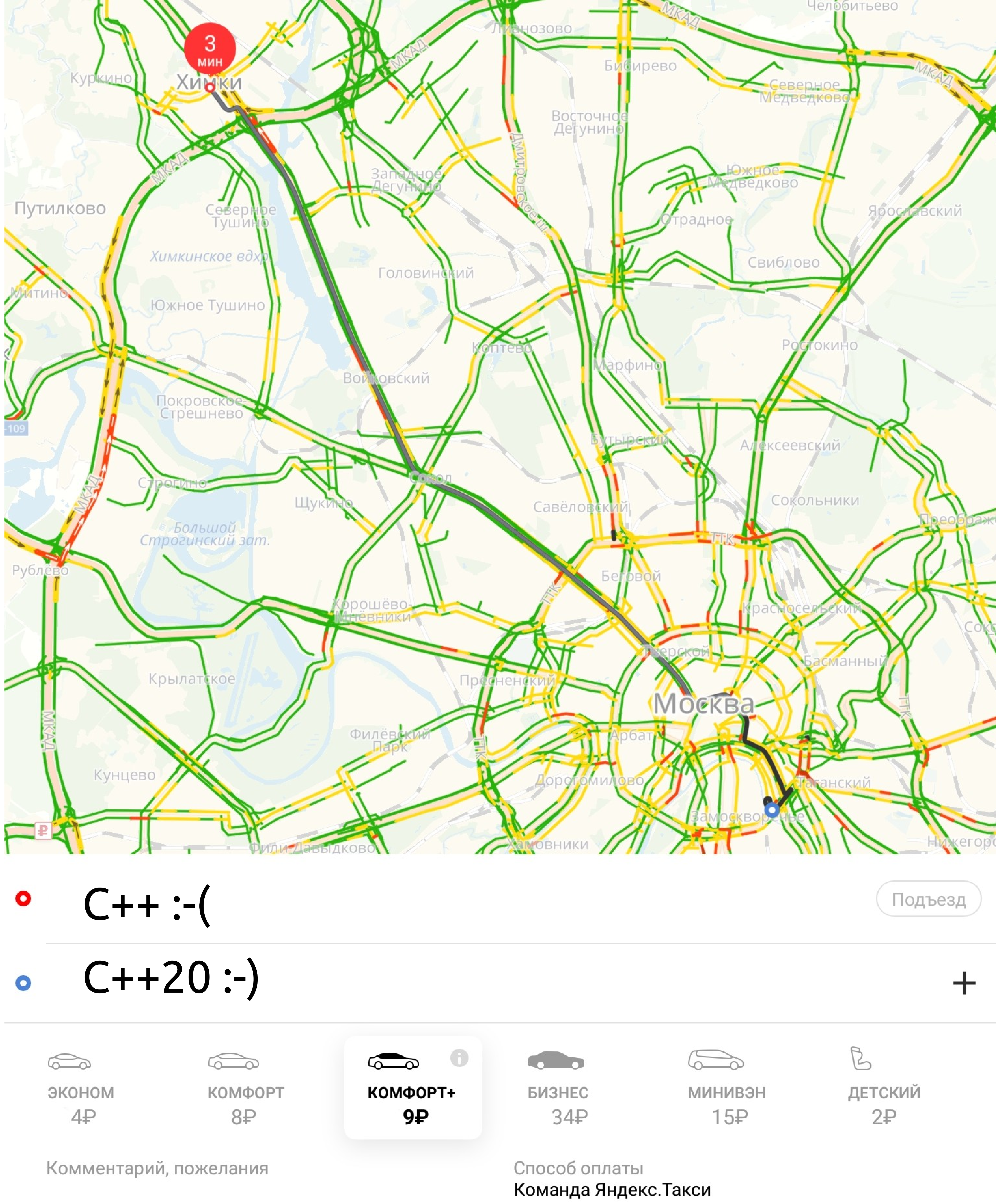
Полухин Антон

Antony Polukhin

Яндекс Go

Содержание

- std::format
- Концепты
- Ranges
- Календарь
- Aggregate variable(Args...);
- Thread
- Modules
- Coroutines
- C++17
- Комипляторы



C++ :-(
C++20 :-)

std::format

std::format

std::format

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

std::format

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


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

std::format

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


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

std::format

```
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::format

```
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::format

```
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);
```

std::format

```
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);
```

std::format

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

std::format

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

std::format (C++23 bugfix)

```
std::format("{0:{1}.{2}f}", 12.345678);
```

std::format (C++23 bugfix)

```
std::format("{0:{1}.{2}f}", 12.345678);
```

```
// error:
//  return std::format("{0:{1}.{2}f}", 12.345678);
//                     ^
```

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

```
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);
}
```

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);
}
```

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);
}
```

Ranges

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  //
    );
  }
```

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
}
```

Aggregate (...)

Aggreagates

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};
```

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};

auto v = std::make_unique<i_am_an_aggreagate>(42, "Hello");
```

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};

auto v = std::make_unique<i_am_an_aggreagate>(42, "Hello");

std::set<i_am_an_aggreagate> s;
s.emplace(42, "Hello");
```

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};

auto v = std::make_unique<i_am_an_aggreagate>(42, "Hello");  // C++17 error

std::set<i_am_an_aggreagate> s;
s.emplace(42, "Hello"); // C++17 error: no matching constructor
```

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};

auto v = std::make_unique<i_am_an_aggreagate>(42, "Hello");  // C++20 OK

std::set<i_am_an_aggreagate> s;
s.emplace(42, "Hello"); // C++20 OK
```

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};

auto v = std::make_unique<i_am_an_aggreagate>(42, "Hello");

std::set<i_am_an_aggreagate> s;
s.emplace(42, "Hello");

i_am_an_aggreagate a(42, "Hello");
```

Aggreagates

```
struct i_am_an_aggreagate {
  int i;
  std::string s;
};

auto v = std::make_unique<i_am_an_aggreagate>(42, "Hello");

std::set<i_am_an_aggreagate> s;
s.emplace(42, "Hello");

i_am_an_aggreagate a(42, "Hello");  // C++17 error, C++20 OK
```

Thread

Новенькое

Новенькое

- std::jthread

Новенькое

- std::jthread

```
std::jthread t([](std::stop_token st) {
  while (!st.stop_requested()) {
    /* ... */
  }
});

/* ... */

t.request_stop();
```

Новенькое

- std::jthread

```
std::jthread t([](std::stop_token st) {
  while (!st.stop_requested()) {
    /* ... */
  }
});

/* ... */

t.request_stop();
```

Новенькое

- std::jthread

```
std::jthread t([](std::stop_token st) {
  while (!st.stop_requested()) {
    /* ... */
  }
});

/* ... */

t.request_stop();
```

Новенькое

- std::jthread
- std::semaphore

Новенькое

- std::jthread
- std::semaphore
- std::latch, std::barrier

Новенькое

- std::jthread
- std::semaphore
- std::latch, std::barrier
- std::atomic::wait

Новенькое

- std::jthread
- std::semaphore
- std::latch, std::barrier
- std::atomic::wait
- std::atomic_ref

Модули

Modules Inro

```
// экспортирует макросы, экспортирует все символы, чувствительно к макросам

#include <iostream>
```

Modules Inro

```
// экспортирует макросы, экспортирует все символы, чувствительно к макросам
#include <iostream>


// экспортирует макросы, экспортирует все символы, к макросам НЕ чувствительно
import <iostream>;
```

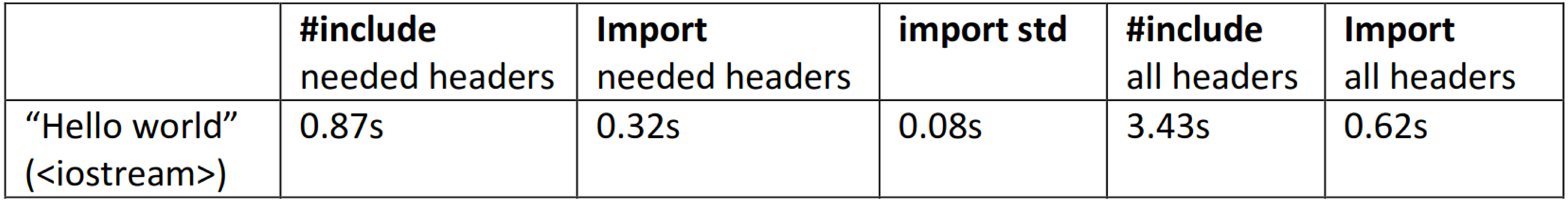
Modules Inro

```
// экспортирует макросы, экспортирует все символы, чувствительно к макросам
#include <iostream>


// экспортирует макросы, экспортирует все символы, к макросам НЕ чувствительно
import <iostream>;


// НЕ экспортирует макросы, к макросам НЕ чувствительно
import my_project.iostream; // TODO: std.iostream не в C++20
```

Модуль std: C++23



	#include needed headers	**Import** needed headers	**import std**	**#include** all headers	**Import** all headers
“Hello world” (<iostream>)	0.87s	0.32s	0.08s	3.43s	0.62s

Корутины

Coroutines links

Coroutines links

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

Coroutines links

- ASIO/Boost.ASIO
- CppCoro https://github.com/lewissbaker/cppcoro
- userver C++ Framework

C++17

C++17

C++17

- std::optional

C++17

- std::optional
- std::variant

C++17

- std::optional
- std::variant
- std::to_chars / std::from_chars

C++17

- std::optional
- std::variant
- std::to_chars / std::from_chars
- auto [it, ok] = map.emplace("key", "value");

C++17

- std::optional
- std::variant
- std::to_chars / std::from_chars
- auto [it, ok] = map.emplace("key", "value");
- std::string_view

C++17

- std::optional
- std::variant
- std::to_chars / std::from_chars
- auto [it, ok] = map.emplace("key", "value");
- std::string_view
- std::filesystem::*

C++17

- std::optional
- std::variant
- std::to_chars / std::from_chars
- auto [it, ok] = map.emplace("key", "value");
- std::string_view
- std::filesystem::*
- if constexpr

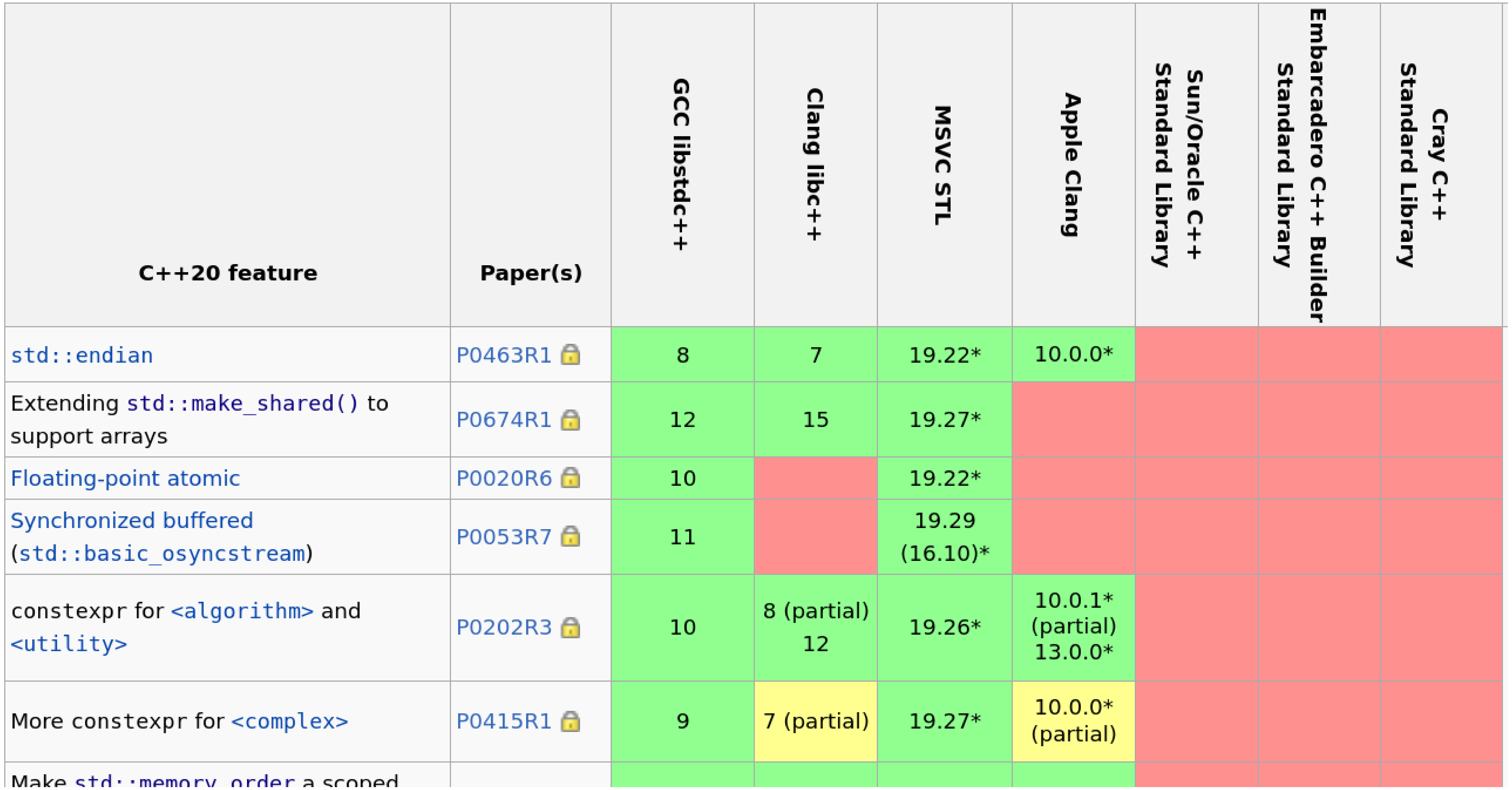
C++17

- std::optional
- std::variant
- std::to_chars / std::from_chars
- auto [it, ok] = map.emplace("key", "value");
- std::string_view
- std::filesystem::*
- if constexpr
- Class template argument deduction
  std::unique_lock lock{some_mutex}

Компиляторы

Компиляторы

https://en.cppreference.com/w/cpp/compiler_support



C++20 feature	Paper(s)	GCC libstdc++	Clang libc++	MSVC STL	Apple Clang	Sun/Oracle C++ Standard Library	Embarcadero C++ Builder Standard Library	Cray C++ Standard Library
std::endian	P0463R1	8	7	19.22*	10.0.0*	🟥	🟥	🟥
Extending std::make_shared() to support arrays	P0674R1	12	15	19.27*	🟥	🟥	🟥	🟥
Floating-point atomic	P0020R6	10	🟥	19.22*	🟥	🟥	🟥	🟥
Synchronized buffered (std::basic_osyncstream)	P0053R7	11	🟥	19.29 (16.10)*	🟥	🟥	🟥	🟥
constexpr for <algorithm> and <utility>	P0202R3	10	8 (partial) 12	19.26*	10.0.1* (partial) 13.0.0*	🟥	🟥	🟥
More constexpr for <complex>	P0415R1	9	7 (partial)	19.27*	10.0.0* (partial)	🟥	🟥	🟥
Make std::memory_order a scoped		🟩	🟩	🟩	🟩	🟥	🟥	🟥

Спасибо

Полухин Антон

Эксперт-разработчик C++

antoshkka@gmail.com

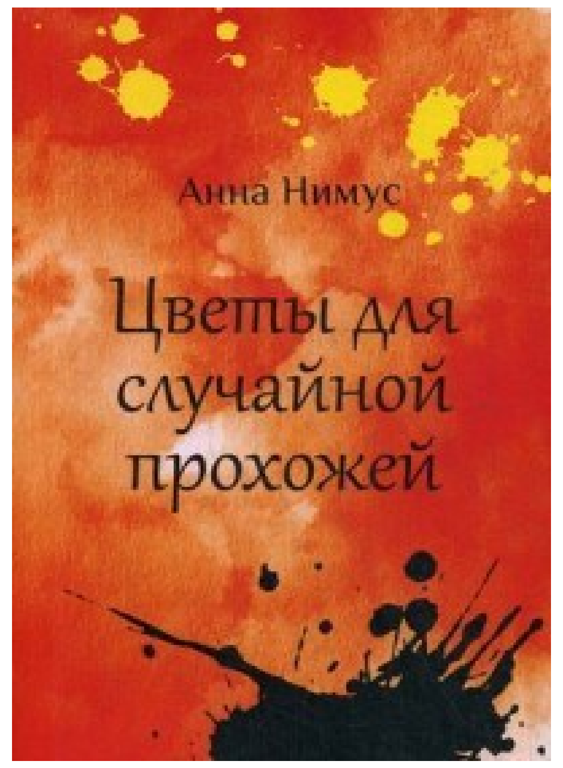
antoshkka@yandex-team.ru

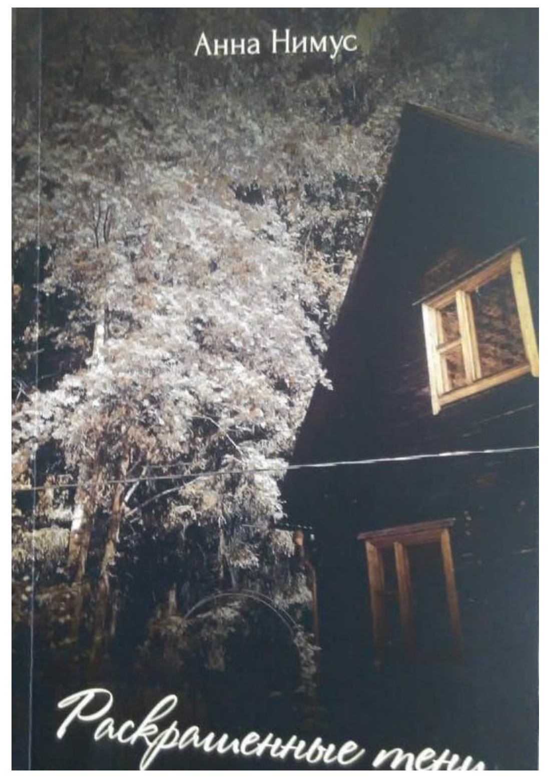
https://github.com/apolukhin

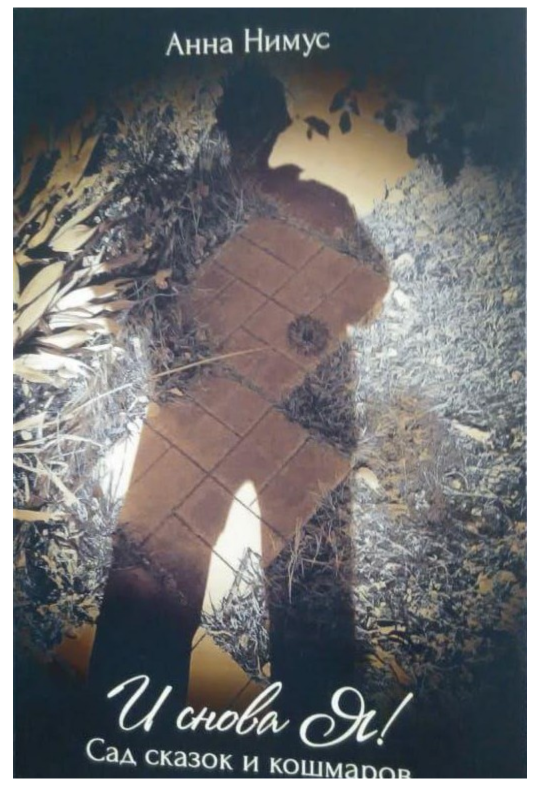
РГ21 C++ РОССИЯ https://stdcpp.ru/

Антон Полухин
Разработка приложений на C++ с использованием Boost
Рецепты, упрощающие разработку вашего приложения
ДМК

Спасибо


Анна Нимус
Цветы для случайной прохожей


Анна Нимус
Раскрашенные тени


Анна Нимус
И снова Я!
Сад сказок и кошмаров