C++23, C++26

Новости последних встреч ISO

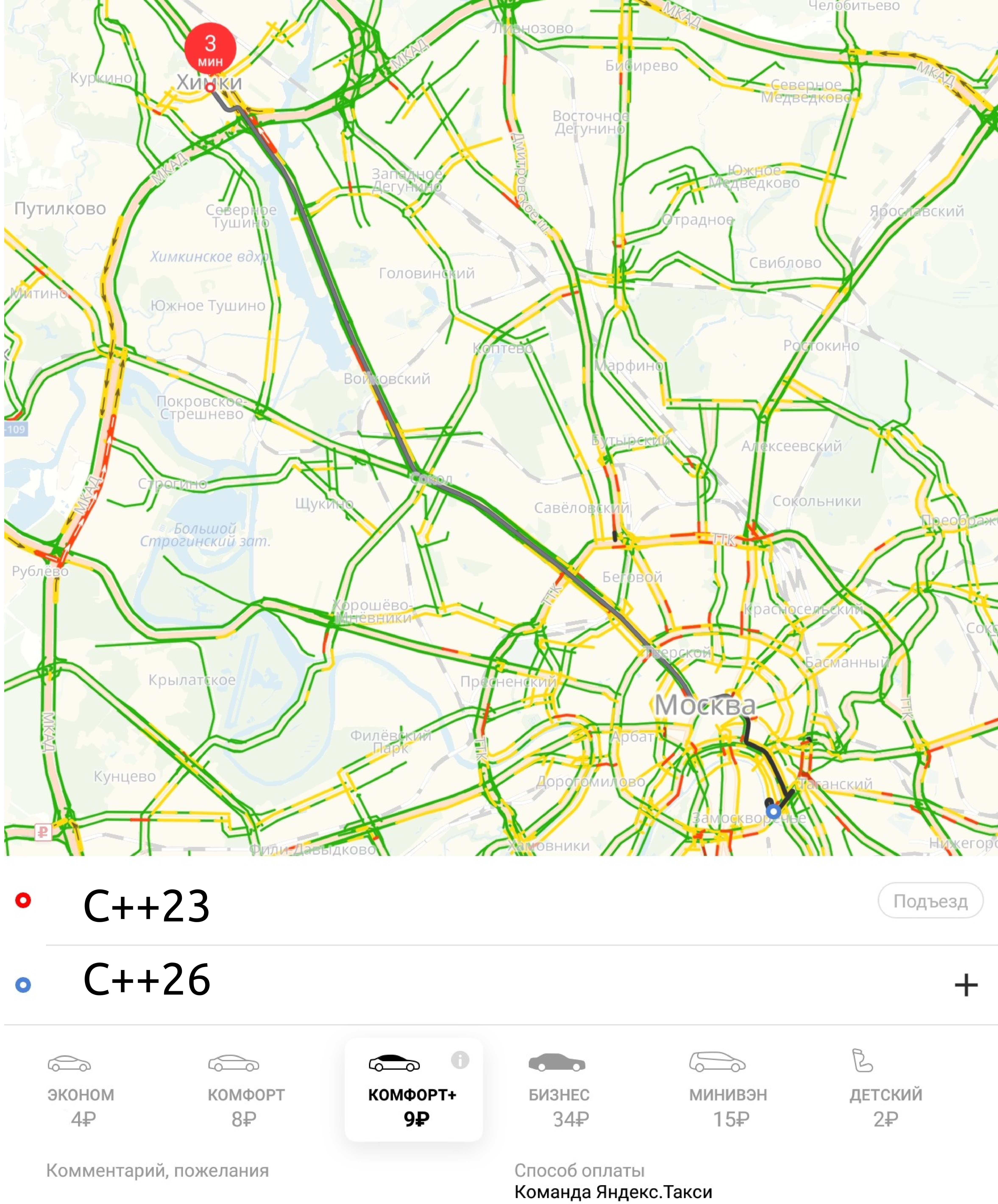
Полухин Антон

Antony Polukhin

РГ21 С++ РОССИЯ

Содержание

- C++23, недавнее:
  - static_assert(false)
  - Безопасный range based for
  - static operator[]
  - std::expected
- C++23
  - ranges
  - std::stacktrace, std::format и std::print
  - constexpr
- C++26



C++23

Подъезд

C++26

+

ЭКОНОМ	КОМФОРТ	КОМФОРТ+	БИЗНЕС	МИНИВЭН	ДЕТСКИЙ
4₽	8₽	9₽	34₽	15₽	2₽

Комментарий, пожелания

Способ оплаты
Команда Яндекс.Такси

static_assert(false)

C++23, C++26

static_assert(false)

static_assert(false)

```
template <class T>
int foo() {








}
```

static_assert(false)

```
template <class T>
int foo() {
    if constexpr (std::is_same_v<T, int>) {
      return 42;






}
```

static_assert(false)

```cpp
template <class T>
int foo() {
    if constexpr (std::is_same_v<T, int>) {
      return 42;
    } else if constexpr (std::is_same_v<T, float>) {
      return 24;



}
```

static_assert(false)

```
template <class T>
int foo() {
    if constexpr (std::is_same_v<T, int>) {
      return 42;
    } else if constexpr (std::is_same_v<T, float>) {
      return 24;
    } else {
      static_assert(false, "T should be an int or a float");
    }
}
```

static_assert(false)

```
template <class T>
int foo() {
    if constexpr (std::is_same_v<T, int>) {
      return 42;
    } else if constexpr (std::is_same_v<T, float>) {
      return 24;
    } else {
      static_assert(false, "T should be an int or a float");
    }
}
```

Безопасный range based for

C++23, C++26

Безопасный range based for

Безопасный range based for

```
class SomeData {
 public:
  // ...


 private:
  std::vector<int> data_;
};
```

Безопасный range based for

```
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};
```

Безопасный range based for

```
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};
```

Безопасный range based for

```
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};
```

Безопасный range based for

```
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};

SomeData Foo();
```

Безопасный range based for

```
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};

SomeData Foo();

int main() {
  for (int v: Foo().Get()) {
    std::cout << v << ',';
  }
}
```

Безопасный range based for

```
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};

SomeData Foo();

int main() {
  for (int v: Foo().Get()) {
    std::cout << v << ',';
  }
}
```

Безопасный range based for

```cpp
class SomeData {
 public:
  // ...
  const std::vector<int>& Get() const { return data_; }

 private:
  std::vector<int> data_;
};

SomeData Foo();

int main() {
  for (int v: Foo().Get()) {
    std::cout << v << ',';
  }
}
```

Безопасный range based for

Безопасный range based for

```
int main() {
  for (int v: Foo().Get()) {
    std::cout << v << ',';
  }
}
```

Безопасный range based for

```
int main() {

  auto && __range = Foo().Get();









}
```

Безопасный range based for

```
int main() {

  auto && __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {


  }
}
```

Безопасный range based for

```
int main() {

  auto && __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {
      int v = *__begin;
      std::cout << v << ',';
  }
}
```

Безопасный range based for

```
int main() {

  auto && __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {
      int v = *__begin;
      std::cout << v << ',';
  }
}
```

Безопасный range based for

```
int main() {

  const std::vector<int>& __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {
      int v = *__begin;
      std::cout << v << ',';
  }
}
```

Безопасный range based for

```
int main() {

  const std::vector<int>& __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {
      int v = *__begin;
      std::cout << v << ',';
  }
}
```

Безопасный range based for

```
int main() {

  const std::vector<int>&  __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {
      int v = *__begin;
      std::cout << v << ',';
  }
}
```

Безопасный range based for

```
int main() {

  const std::vector<int>&  __range = Foo().Get();

  for (auto __begin = __range.begin(), __end = __range.end();
        __begin != __end;
        ++__begin
  ) {
      int v = *__begin;
      std::cout << v << ',';
  }
}
```

static operator[]

C++23, C++26

static operator[]

static operator[]

```
enum class Color { red, green };
```

static operator[]

```
enum class Color { red, green };

struct kEnumToStringViewBimap {
```

static operator[]

```
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept {
    switch(color) {
    case Color::red: return "red";
    case Color::green: return "green";
    }
  }
```

static operator[]

```
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept {
    switch(color) {
    case Color::red: return "red";
    case Color::green: return "green";
    }
  }
```

static operator[]

```cpp
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept {
    switch(color) {
    case Color::red: return "red";
    case Color::green: return "green";
    }
  }
```

static operator[]

```cpp
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept {
    switch(color) {
    case Color::red: return "red";
    case Color::green: return "green";
    }
  }

  static constexpr Color operator[](std::string_view color) noexcept {
    if (color == "red") {
      return Color::red;
    } else if (color == "green") {
      return Color::green;
    }
  }
};
```

static operator[]

static operator[]

```cpp
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept;
  static constexpr Color operator[](std::string_view color) noexcept;
};

static_assert(kEnumToStringViewBimap["red"] == Color::red);
```

static operator[]

```cpp
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept;
  static constexpr Color operator[](std::string_view color) noexcept;
};

static_assert(kEnumToStringViewBimap["red"] == Color::red);
```

static operator[]

```cpp
enum class Color { red, green };

struct kEnumToStringViewBimap {
  static constexpr std::string_view operator[](Color color) noexcept;
  static constexpr Color operator[](std::string_view color) noexcept;
};

static_assert(kEnumToStringViewBimap["red"] == Color::red);
```

Монадические операции std::expected

C++23, C++26

std::expected

std::expected

```
using std::chrono::system_clock;
```

std::expected

```
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
```

std::expected

```
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
```

std::expected

```cpp
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);
```

std::expected

```
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
```

std::expected

```
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
from_iso_str(input_data)
```

std::expected

```cpp
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
from_iso_str(input_data)
    .and_then(&to_bson)
```

std::expected

```
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
from_iso_str(input_data)
    .and_then(&to_bson)
    .and_then(&insert_into_db)
```

std::expected

```cpp
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
from_iso_str(input_data)
    .and_then(&to_bson)
    .and_then(&insert_into_db)
    .transform_error([](std::string_view error) -> std::string_view {
        throw Exception(error);
    })
;
```

std::expected

```
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
from_iso_str(input_data)
    .and_then(&to_bson)
    .and_then(&insert_into_db)
    .transform_error([](std::string_view error) -> std::string_view {
        throw Exception(error);
    })
;
```

std::expected

```cpp
using std::chrono::system_clock;

std::expected<system_clock, std::string> from_iso_str(std::string_view time);
std::expected<formats::bson::Timestamp, std::string> to_bson(system_clock time);
std::expected<int, std::string> insert_into_db(formats::bson::Timestamp time);

// Где-то в коде приложения...
from_iso_str(input_data)
    .and_then(&to_bson)
    .and_then(&insert_into_db)
    .transform_error([](std::string_view error) -> std::string_view {
        throw Exception(error);
    })
;
```

C++23

C++23, клёвые штуки

Ranges

C++23, C++26

Ranges

Ranges

```
std::map<std::string, int> m;
```

Ranges

```
std::map<std::string, int> m;

for (const auto& chunk : m
     | std::views::keys
```

Ranges

```
std::map<std::string, int> m;

for (const auto& chunk : m
     | std::views::keys
     | std::views::take(5)
```

Ranges

```
std::map<std::string, int> m;

for (const auto& chunk : m
     | std::views::keys
     | std::views::take(5)
     | std::views::join_with(", ")
```

Ranges

```
std::map<std::string, int> m;

for (const auto& chunk : m
     | std::views::keys
     | std::views::take(5)
     | std::views::join_with(", "))
{
  std::cout << chunk;
}
```

Ranges

```
std::map<std::string, int> m;

auto str = m
     | std::views::keys
     | std::views::take(5)
     | std::views::join_with(", ")
     | std::ranges::to<std::string>();
```

std::stacktrace, std::format, std::print

C++23, C++26

Utilities

Utilities

```
void log_error(std::string_view error) {








}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();





}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {

    } else {

    }
}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error);
    }
}
```

Utilities

```cpp
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error);
    }
}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error);
    }
}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error);
    }
}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error, trace);
    }
}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error, trace); // Compile time error
    }
}
```

Utilities

```
void log_error(std::string_view error) {
    auto trace = std::stacktrace::current();
    if (trace) {
        std::print("Error '{}' at:\n{}", error, trace);
    } else {
        std::print("Error '{}'", error, trace); // Compile time error
    }
}
```

constexpr

C++23, C++26

C++26

P2000

P2000

- Library support for coroutines

P2000

- Library support for coroutines
- Executors

P2000

- Library support for coroutines
- Executors
- Networking

P2000

- Library support for coroutines
- Executors
- Networking

- Pattern Matching

P2000

- Library support for coroutines
- Executors
- Networking

- Pattern Matching
- Reflection

P2000

- Library support for coroutines
- Executors
- Networking

- Pattern Matching
- Reflection

- Transaction

P2000

- Library support for coroutines
- Executors
- Networking

- Pattern Matching
- Reflection

- Transaction
- Freestanding

P2000

- Library support for coroutines
- Executors
- Networking

- Pattern Matching
- Reflection

- Transaction
- Freestanding
- Better support for C++ ecosystem

C++26

#embed

#embed

```
const std::byte icon_display_data[] = {
    #embed "art.png"
};
```

C++26 от РГ21

std::get для агрегатов

C++23, C++26

std::get для агрегатов

std::get для агрегатов

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

std::get для агрегатов

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

Aggregate aggr{42, "hello"};
```

std::get для агрегатов

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

Aggregate aggr{42, "hello"};

assert(std::get<0>(aggr) == 42);
assert(std::get<1>(aggr) == "hello");
```

std::get для агрегатов

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

Aggregate aggr{42, "hello"};

assert(std::get<0>(aggr) == 42);
assert(std::get<1>(aggr) == "hello");

static_assert(std::tuple_size_v<Aggregate> == 2);
```

std::get для агрегатов

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

Aggregate aggr{42, "hello"};

assert(std::get<0>(aggr) == 42);
assert(std::get<1>(aggr) == "hello");

static_assert(std::tuple_size_v<Aggregate> == 2);
```

std::get для агрегатов

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

Aggregate aggr{42, "hello"};

assert(std::get<0>(aggr) == 42);
assert(std::get<1>(aggr) == "hello");

static_assert(std::tuple_size_v<Aggregate> == 2);
```

std::get для агрегатов

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

Aggregate aggr{42, "hello"};

assert(std::get<0>(aggr) == 42);
assert(std::get<1>(aggr) == "hello");

~~static_assert(std::tuple_size_v<Aggregate> == 2);~~
static_assert(std::elements_count_v<Aggregate> == 2);
```

Stacktrace from exception

C++23, C++26

std::stacktrace из исключения

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {


  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::cerr << "Caught exception: " << exc.what();

  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::cerr << "Caught exception: " << exc.what();

  }
}
```

std::stacktrace из исключения

```cpp
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::cerr << "Caught exception: " << exc.what();

  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::cerr << "Caught exception: " << exc.what();  // Caught exception: map::at

  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {


  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::stacktrace trace = std::stacktrace::from_current_exception();

  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::stacktrace trace = std::stacktrace::from_current_exception();

  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::stacktrace trace = std::stacktrace::from_current_exception();

  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::stacktrace trace = std::stacktrace::from_current_exception();
    std::cerr << "Caught exception: " << exc.what() << ", trace:\n" << trace;
  }
}
```

std::stacktrace из исключения

```
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::stacktrace trace = std::stacktrace::from_current_exception();
    std::cerr << "Caught exception: " << exc.what() << ", trace:\n" << trace;
  }
}
```

std::stacktrace из исключения

```cpp
#include <iostream>
#include <stdexcept>
#include <string_view>
#include <stacktrace>

void foo(std::string_view key);
void bar(std::string_view key);

int main() {
  try {
    foo("test1");
    bar("test2");
  } catch (const std::exception& exc) {
    std::stacktrace trace = std::stacktrace::from_current_exception();
    std::cerr << "Caught exception: " << exc.what() << ", trace:\n" << trace;
  }
}
```

Спасибо

Полухин Антон

Эксперт-разработчик C++

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/

РГ21 C++ РОССИЯ