



*Spring 2021*

*[compscicenter.ru](http://compscicenter.ru)*

Basharin Egor

[t.me/egorbasharin](https://t.me/egorbasharin)

# Lecture IX

C++20 Coroutines

# Section 0

warm up

# Организационные вопросы

- Формирование оценки: [materials](#)
- Видео семинаров: [#lecture](#)

# shared\_ptr destructor

```
struct T {  
    ~T() { std::cout << 1; }  
};  
struct D : T {  
    ~D() { std::cout << 2; }  
};  
  
int main() {  
    std::vector<std::shared_ptr<T>> vs;  
    vs.emplace_back(new D);  
    // вызовется ли деструктор D  
  
    return 0;  
}
```

[Click me](#)

# serialization

Идеи с сериализацией/десериализацией

<https://gist.github.com/eaniconer/003cb206e1b58429a8bb1f1f8519c38>

# Section 1

Motivation

# Example 1

```
task<int> coroutine() {
    T t;
    std::cout << "step1" << std::endl;
    int a = 1;
    co_await coro::suspend_always{};
    std::cout << "step2" << std::endl;
    int b = 2;
    co_await coro::suspend_always{};
    std::cout << "step3" << std::endl;
    co_return a + b;
}

int main() {
    [[maybe_unused]] auto task = coroutine();
    std::cout << "WATCHPOINT1" << std::endl; task.resume();
    std::cout << "WATCHPOINT2" << std::endl; task.resume();
    std::cout << "WATCHPOINT3" << std::endl; task.resume();
    std::cout << "WATCHPOINT4" << std::endl;
}
```



# Example 2

```
generator<int> range(size_t n) {  
    for (size_t i = 0; i < n; ++i) {  
        co_yield i;  
    }  
}  
  
int main() {  
    for (auto i : range(10)) {  
        std::cout << i << " ";  
    }  
}
```

# A coroutine

*A coroutine is a function that can suspend execution to be resumed later.*

# C++ coroutines

- stackless
- lightweight
- no context switches
- cooperative multitasking
- memory allocations are allowed

# C++ coroutines

Стандарт не предоставляет корутин.

Стандарт предоставляет фреймворк для их создания:

- сложность (написания, понимания)
- эффективность
- тонкая настройка

# C++ coroutines

Функция называется корутиной, если она содержит хотя бы одно из ключевых слов:

- `co_return`
- `co_await`
- `co_yield`

# Section 2

My First Coroutine

# My First Coroutine

```
task<int> answer() {  
    int a = 0b101010  
    co_return a;  
}
```

# Coroutine under the hood

```
task<int> answer() {  
    struct Frame {  
        task<int>::promise_type promise;  
        // state: func params, local vars, resume point, ...  
    }* frame = new Frame{};  
  
    void* ret = new char[sizeof(task<int>)];  
    new (ret) task<int>(frame->promise.get_return_object());  
  
    try {  
        co_await frame->promise.initial_suspend();  
        int a = 0b101010;  
        frame->promise.return_value(a);  
    } catch (...) {  
        frame->promise.unhandled_exception();  
    }  
    co_await frame->promise.final_suspend();  
}
```



# Generator under the hood

```
generator<int> range(size_t n) {  
    for (size_t i = 0; i < n; ++i) {  
        co_yield i; // co_await promise.yield_value(i);  
    }  
    // promise.return_void()  
}
```

# Section 3

`co_await Awaitable`

# Example I

```
using Awaitable = ...;
task<void> step_by_step() {
    std::cout << "step1\n";
    co_await Awaitable{};
    std::cout << "step2\n";
}

int main() {
    auto task = step_by_step();
    std::cout << "watch1\n";
    task.resume();
    std::cout << "watch2\n";
    task.resume();
    std::cout << "watch3\n";
}
```

# co\_await <expr> under the hood

```
auto&& awaiter = <expr>;  
if (!awaiter.await_ready()) {  
    awaiter.await_suspend(coro_handle);  
    suspend();  
}  
awaiter.await_resume();
```

# Example I

```
task<void> step_by_step() {  
    std::cout << "step1\n";  
  
    auto&& awaiter = Awaitable{};  
    if (!awaiter.await_ready()) {  
        awaiter.await_suspend(  
            coro::coroutine_handle<task<void>::promise_t  
                ::from_promise(frame->promise));  
        suspend();  
    }  
    awaiter.await_resume();  
  
    std::cout << "step2\n";  
}
```

# Example II

```
task<int> do_work() {  
    int res = co_await async_task();  
    co_return res + 2;  
}  
  
int main() {  
    auto task = do_work();  
    std::cout << task.result();  
}
```

# Example III

```
task<int> do_work() {  
    co_await Sleep { std::chrono::seconds {1} };  
    co_return 20;  
}  
  
int main() {  
    auto task = do_work();  
    std::cout << task.result();  
}
```

auto&& r = co\_await <expr>

```
auto&& awaiter = <expr>;  
if (!awaiter.await_ready()) {  
    awaiter.await_suspend(coro_handle);  
    suspend();  
}  
auto&& r = awaiter.await_resume();
```