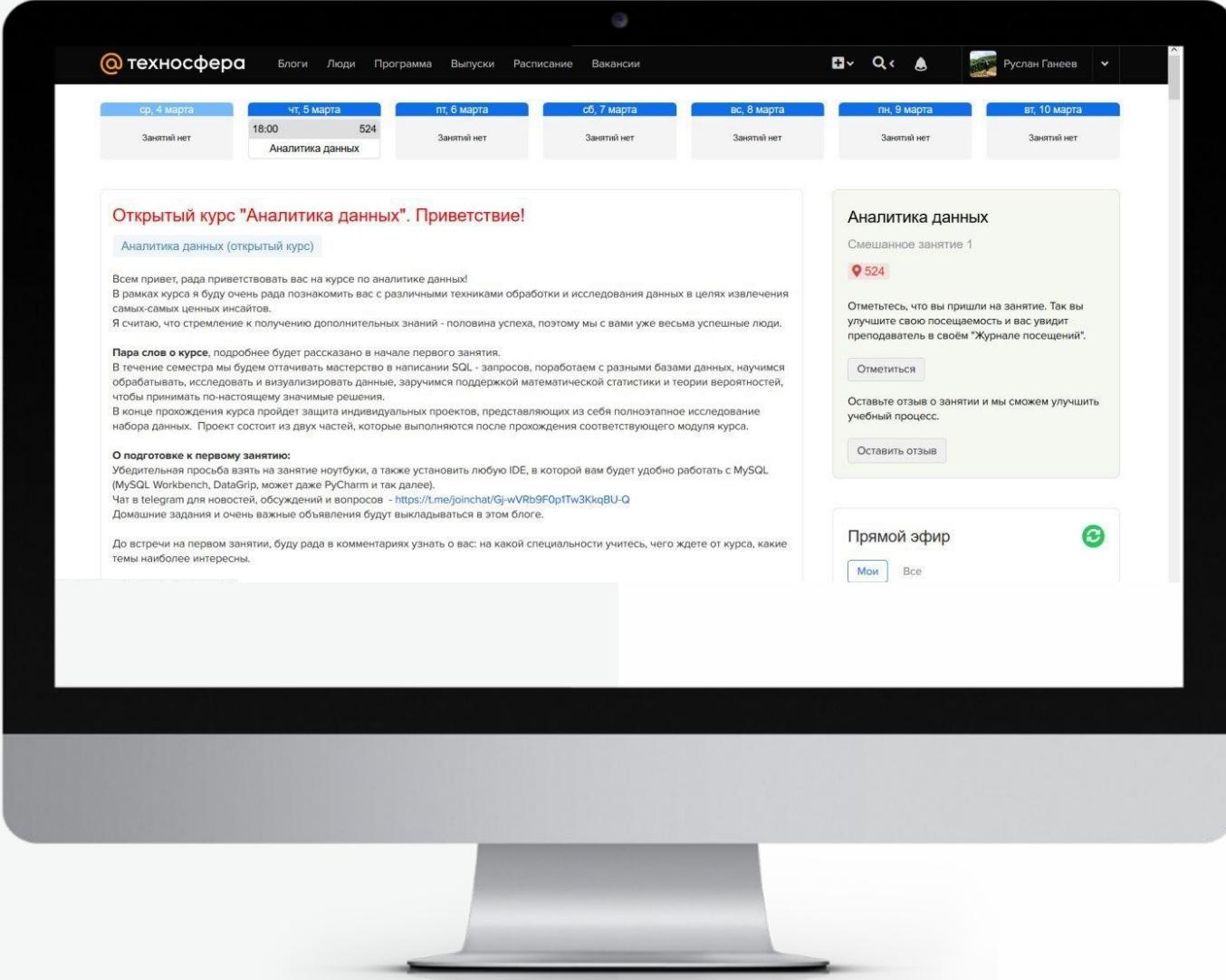




Многопоточность. Часть II

Антон Кухтичев



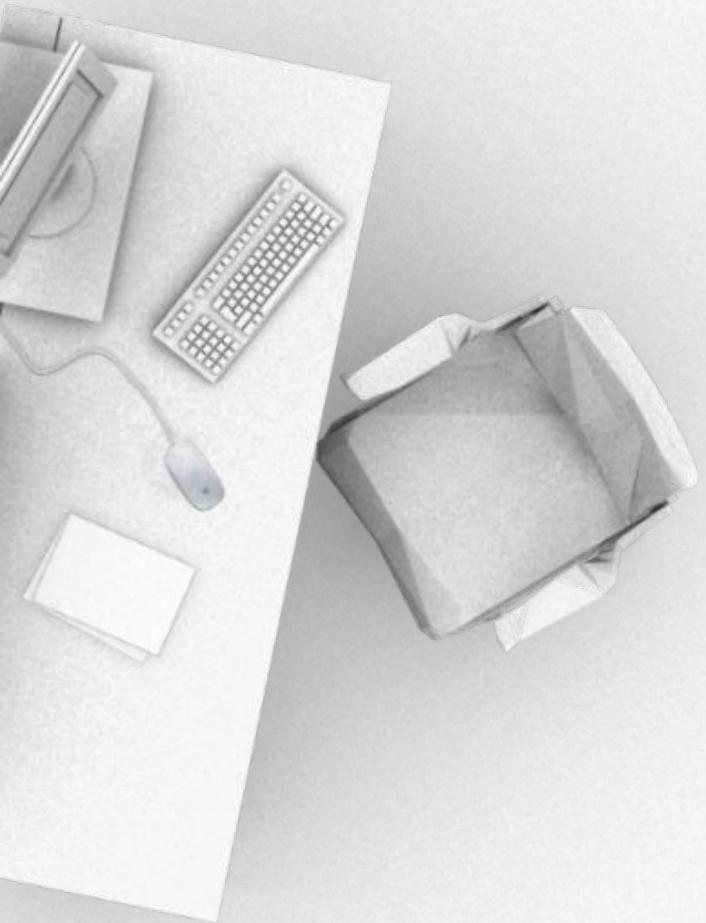


Не забудьте
отметиться на
портале!!!

Иначе всё плохо будет.

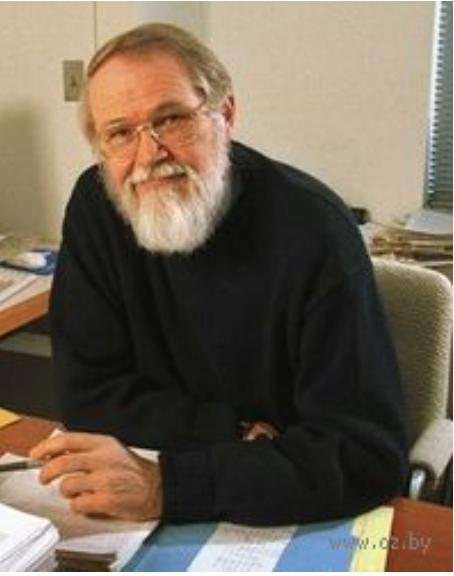
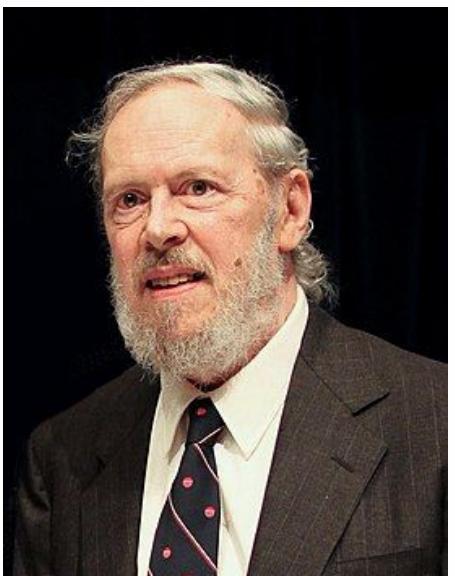
Содержание занятия

1. Задачки
2. Атомарные операции
3. std::shared_mutex
4. std::atomic_flag

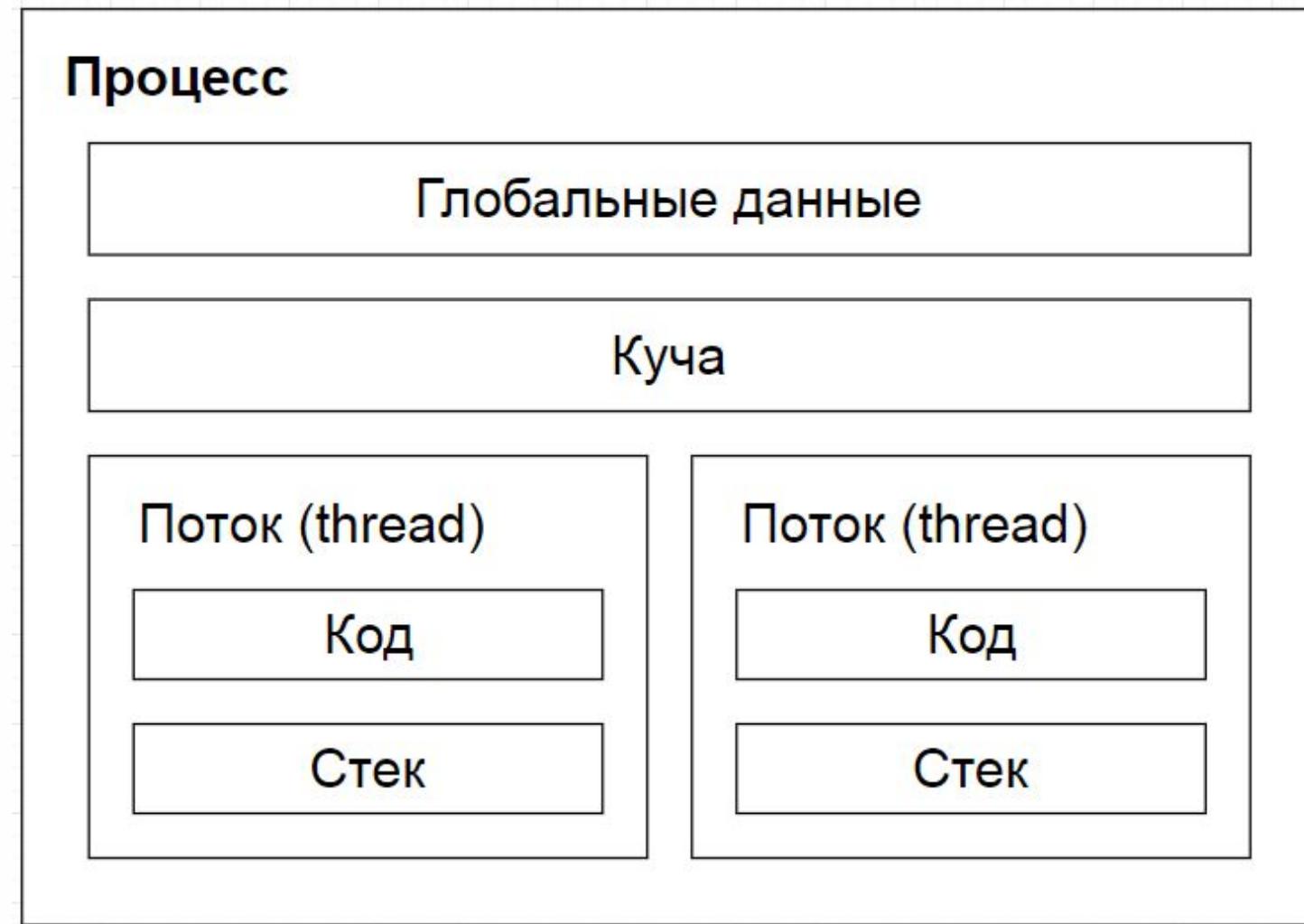


**Угадай
программиста по
фотографии**

Кто это?



Многопоточность (multithreading)



std::call_once

```
std::unique_ptr<Display> display;

void print(const std::string& message)
{
    if (!display)
        display.reset(new Display());
    display->print(message);
}
```

```
std::unique_ptr<Display> display;

static std::once_flag displayInitFlag;

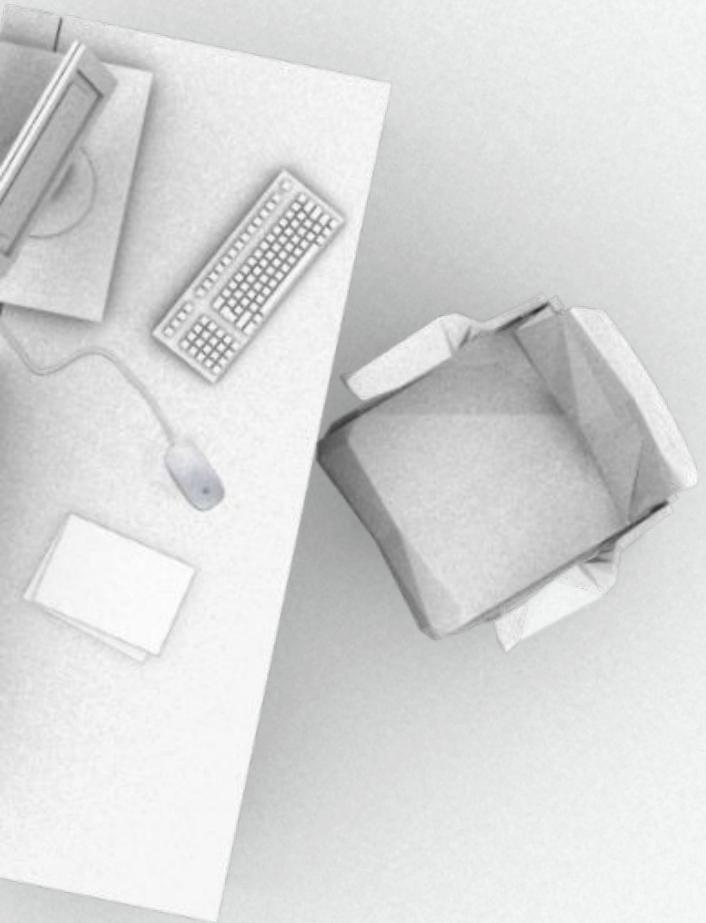
void print(const std::string& message)
{
    std::call_once(displayInitFlag,
                  []() { display.reset(new
Display()); });
    display->print(message);
}
```

thread_local

Хранилище уровня потока (C++11).

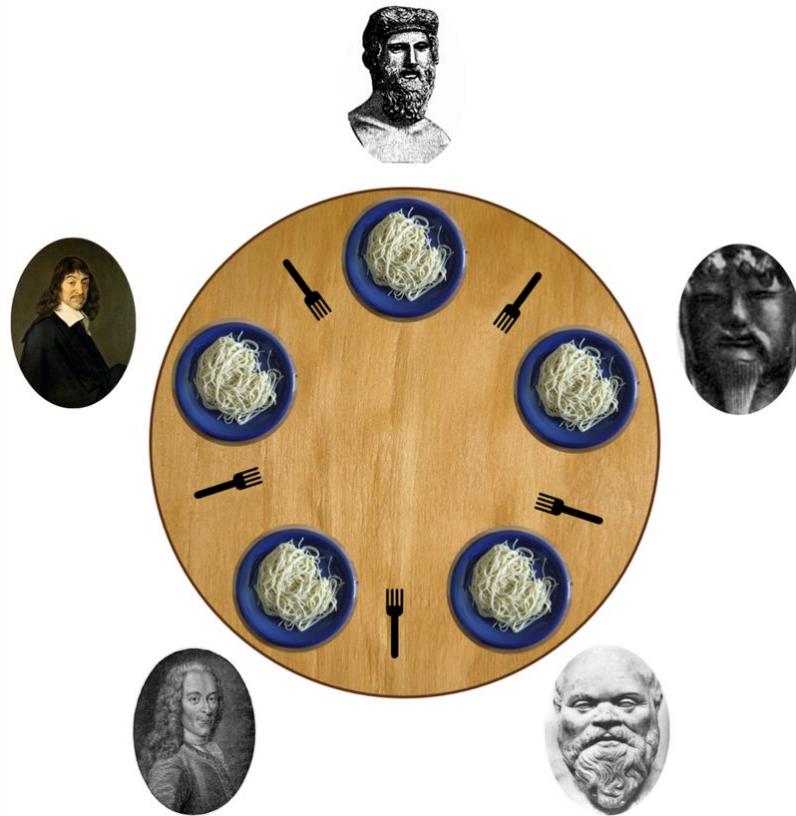
- Создается когда запускается поток
- Уничтожается когда поток завершает работу
- Для каждого потока своё

```
static thread_local std::map<std::string, int> threadCache;
```



Задачки

Задача об обедающих философах



n философов сидят за столом, перед каждым тарелка с едой, между тарелками лежит по 1 палочке. Чтобы поесть, надо взять 2 палочки, то есть философ должен взять палочку слева и справа от своей тарелки.

Задача об обедающих философах

1. Философы не вежливые

Философ взял правую палочку и не отдает, левую палочку взял сосед и тоже не отдает
- произошла взаимная блокировка (deadlock).

2. Философы вежливые

Философ взял правую палочку, посмотрел, что левая занята соседом, положил правую, подумал, повторил с начала. Остальные поступают аналогично. Если все думают одинаковое время, то никто из них не сможет поесть (livelock).

3. Философы слишком быстро едят

Поэтому палочка либо слева, либо справа все время занята и философ голодает (starvation).

Решение

1. Иерархия ресурсов

Надо пронумеровать палочки, далее философы берут палочку с наименьшим номером, затем с наибольшим, возвращают палочки в обратном порядке. Если $n - 1$ философов взяли палочке, то останется одна с наибольшим номером, поэтому ее последний философ взять не сможет. Один из взявших палочку берет палочку с наибольшим номером, затем возвращает ее, позволяя поесть следующему.

Решение

2. Официант

По запросу выдает палочки или предлагает подождать, если все палочки заняты.

Проблема спящего парикмахера

Цель - парикмахер должен работать, когда клиенты есть и спать, когда их нет. Клиент, приходя в парикмахерскую, если парикмахер свободен стрижется, если занят, то идет в приёмную и там садится на стул и ждет, если свободных стульев нет, то уходит.

Producer-consumer problem - частный случай этой задачи.

Code time!



1. Пишем решение задачи о философах
2. Пишем решение задачи о парикмахере

Потокобезопасные интерфейсы

```
template <class T>
class queue
{
    bool empty() const;
    T pop();
};

queue<Task> tasks;

if (!tasks.empty())
    process(tasks.pop());
```

Потокобезопасные интерфейсы

```
template <class T>
class ThreadSafeQueue
{
    bool tryPop(T& value)
    {
        std::lock_guard<std::mutex> lock(mutex);
        if (data_.empty())
            return false;
        value = data_.front();
        data_.pop();
        return true;
    }
};
```

shared_mutex (C++14)

Есть область памяти, позволяющая чтение и запись. Несколько потоков имеют к ней доступ, при этом одновременно могут читать сколько угодно потоков, но писать — только один. Как обеспечить такой режим доступа?

1. Приоритет читателя

Пока память открыта на чтение, давать читателям беспрепятственный доступ.

Писатели могут ждать сколько угодно.

2. Приоритет писателя

Как только появился хоть один писатель, читателей больше не пускать. При этом читатели могут простоять.

3. Одинаковый приоритет

Независимо от действий других потоков, читатель или писатель должен пройти барьер за конечное время.

shared_mutex (C++14)

Класс `std::shared_mutex` - это примитив синхронизации, который может использоваться для защиты общих данных от одновременного доступа нескольких потоков. В отличие от других типов mutex, которые облегчают эксклюзивный доступ, `shared_mutex` имеет два уровня доступа:

- общий - несколько потоков могут совместно владеть одним и тем же mutex .
- эксклюзив - только один поток может владеть mutex.

shared_mutex (C++14)

```
#include <shared_mutex>

std::shared_mutex mutex;

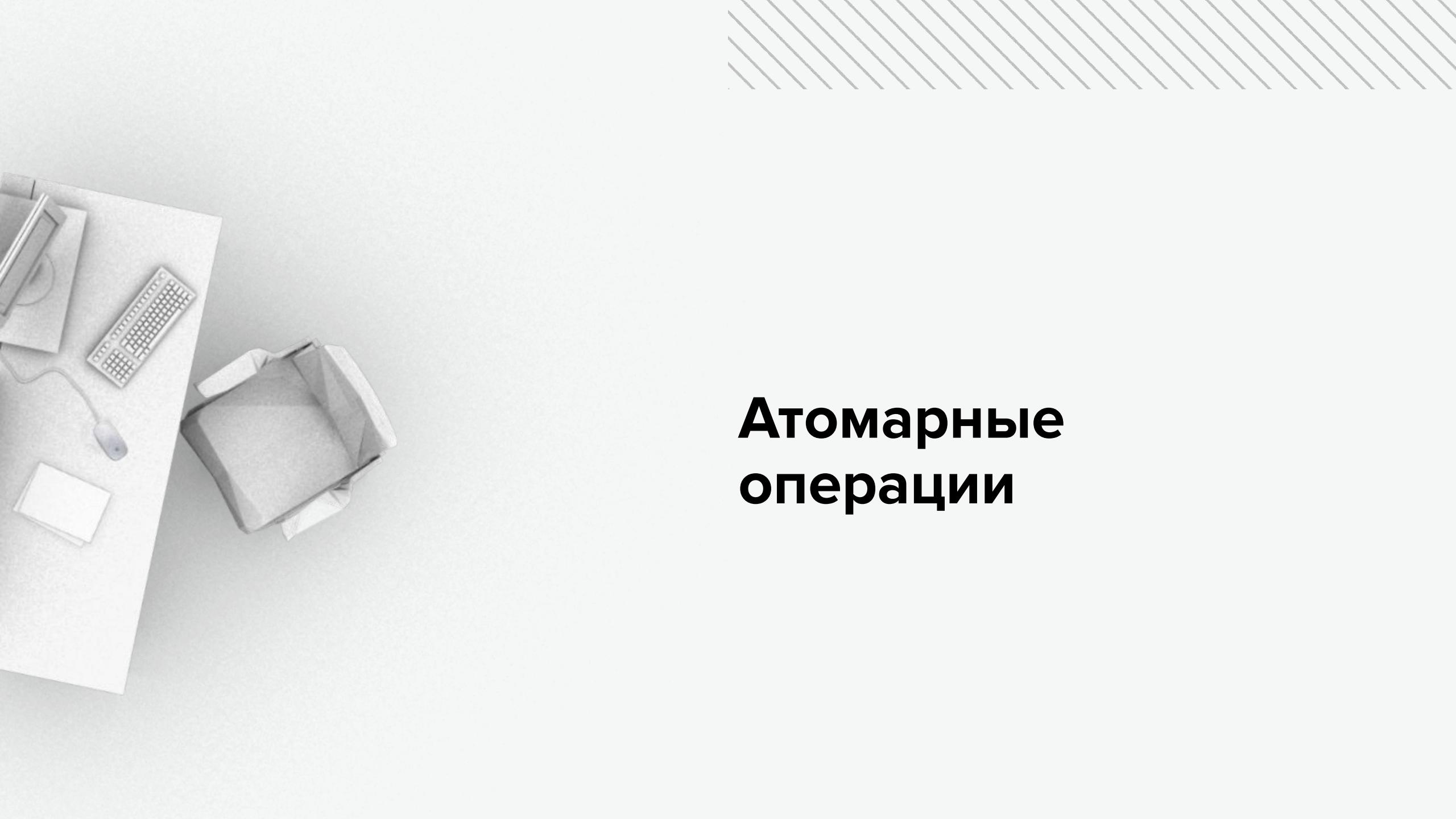
void reader()
{
    std::shared_lock<std::shared_mutex> lock(mutex);
    // блокируется если есть unique_lock
    // не блокируется, если есть другие shared_lock
}

void writer()
{
    std::unique_lock<std::shared_mutex> lock(mutex);
    // получить эксклюзивный доступ на общих условиях
```

Code time!



1. Пишем потокобезопасный кэш



Атомарные операции

std::atomic_flag

std::atomic атомарно изменяет значение, гарантия, что изменение не приводит к блокировке есть только у atomic_flag.

Класс std::atomic_flag предоставляет самый простой атомарный флаг. Это единственный тип данных, гарантированно свободный от блокировок согласно стандарту C++11. Экземпляр std::atomic_flag является либо установленным, либо сброшенным.

std::atomic_flag

```
// флаг сброшен
std::atomic_flag keepWorking = ATOMIC_FLAG_INIT;

// устанавливает флаг и возвращает предыдущее значение
keepWorking.test_and_set();

void threadFunc()
{
    while (keepWorking.test_and_set(std::memory_order_acquire))
    {
    }
}

// сбросить флаг
keepWorking.clear(std::memory_order_release);
```

std::atomic<T>

```
std::atomic<int> i;
```

```
i.is_lock_free(); // true, если изменение не приводит к блокировке
```

```
i.store(5);
```

```
int j = i.exchange(7); // атомарно сохраняет новое значение и  
возвращает предыдущее значение
```

```
int k = 7;
```

```
bool ok = i.compare_exchange_weak(k, 9);
```

```
// атомарно сравнивает значение с ожидаемым значением и сохраняет  
// новое значение, если они равны друг другу и обновление можно  
// выполнить атомарно
```

std::atomic<T>

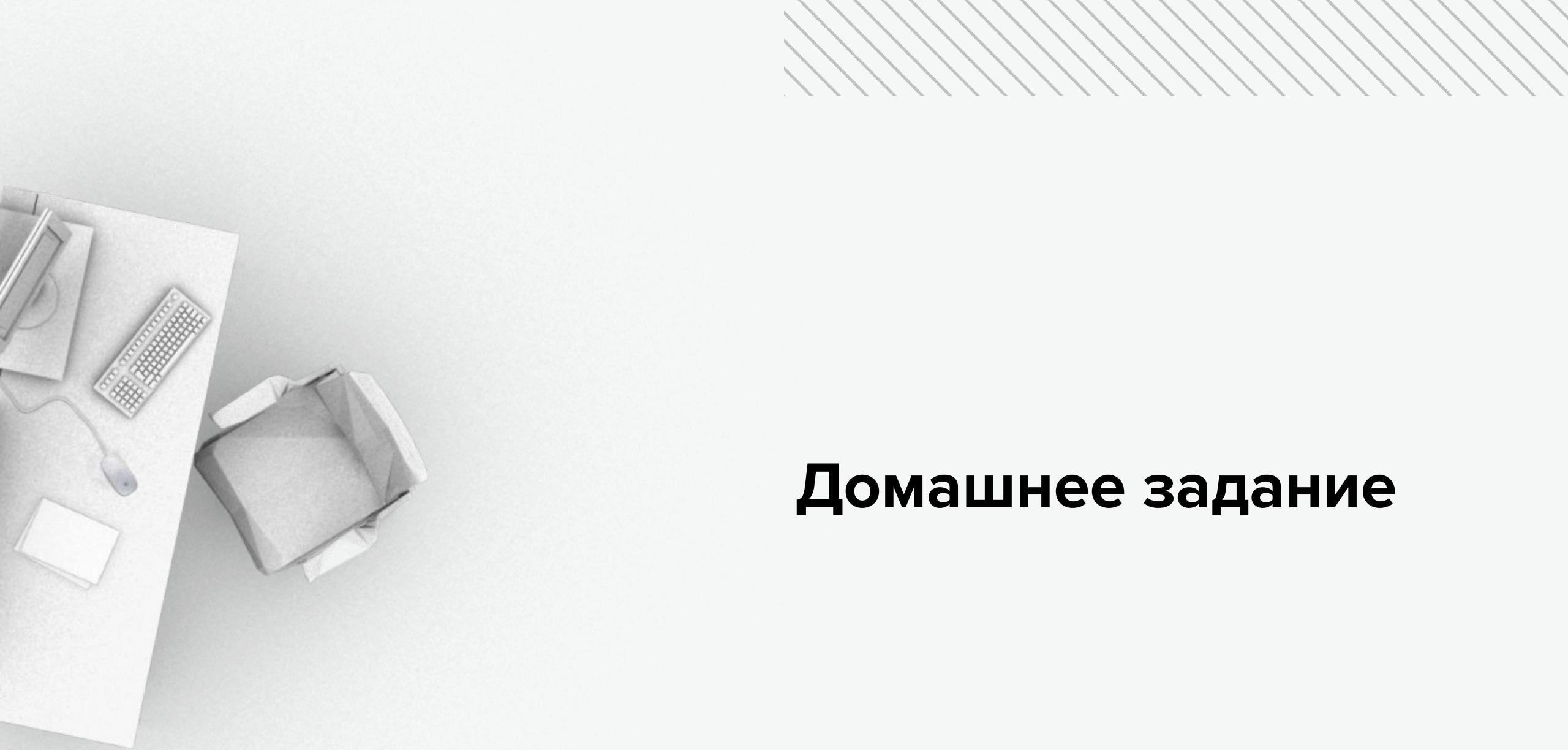
```
ok = i.compare_exchange_strong(k, 9);
// аналогично compare_exchange_weak, но не бывает ложных
срабатываний
// как если бы *this != expected

while (!i.compare_exchange_weak(k, 9));
// compare_exchange_weak можно использовать в цикле, на некоторых
архитектурах
// это дает более эффективный код
```

std::atomic<T>

```
std::atomic<int> i;  
int prev = i.fetch_add(5);
```

```
// fetch_sub  
// fetch_and  
// fetch_or  
// fetch_xor
```



Домашнее задание

Домашнее задание (1)

Выполнить сортировку бинарного файла содержащего числа `uint64_t` в 2 потока.

Доступно 8 Мб памяти, больше выделять нельзя ни явно, ни неявно (например, в виде контейнеров), при этом файл может быть размером превышающим доступную память.

Пространство на диске можно считать бесконечным. Сортировку выполнять в новый файл.

Домашнее задание по уроку #10

Домашнее задание №8

#030

?

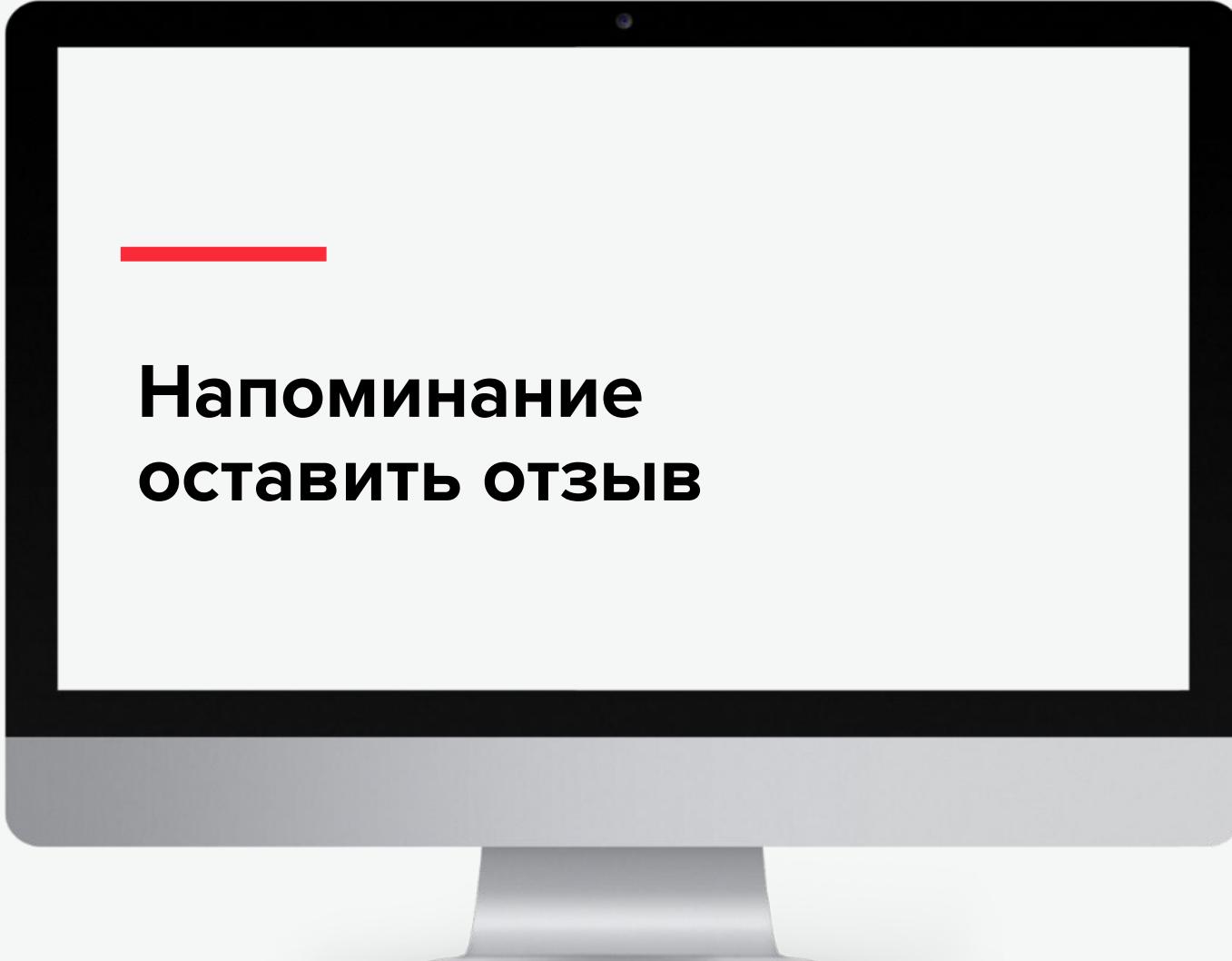
21.05.21

Баллов
за задание

Срок
сдачи

Полезная литература в помощь

- [Документация стандартной библиотеки](#)
- Скотт Мейерс. “Эффективный и современный C++”.
- Энтони Уильямс. “C++. Практика многопоточного программирования”



**Напоминание
оставить отзыв**

**СПАСИБО
ЗА ВНИМАНИЕ**

