Яндекс Такси

Read Me!

Прочти меня!

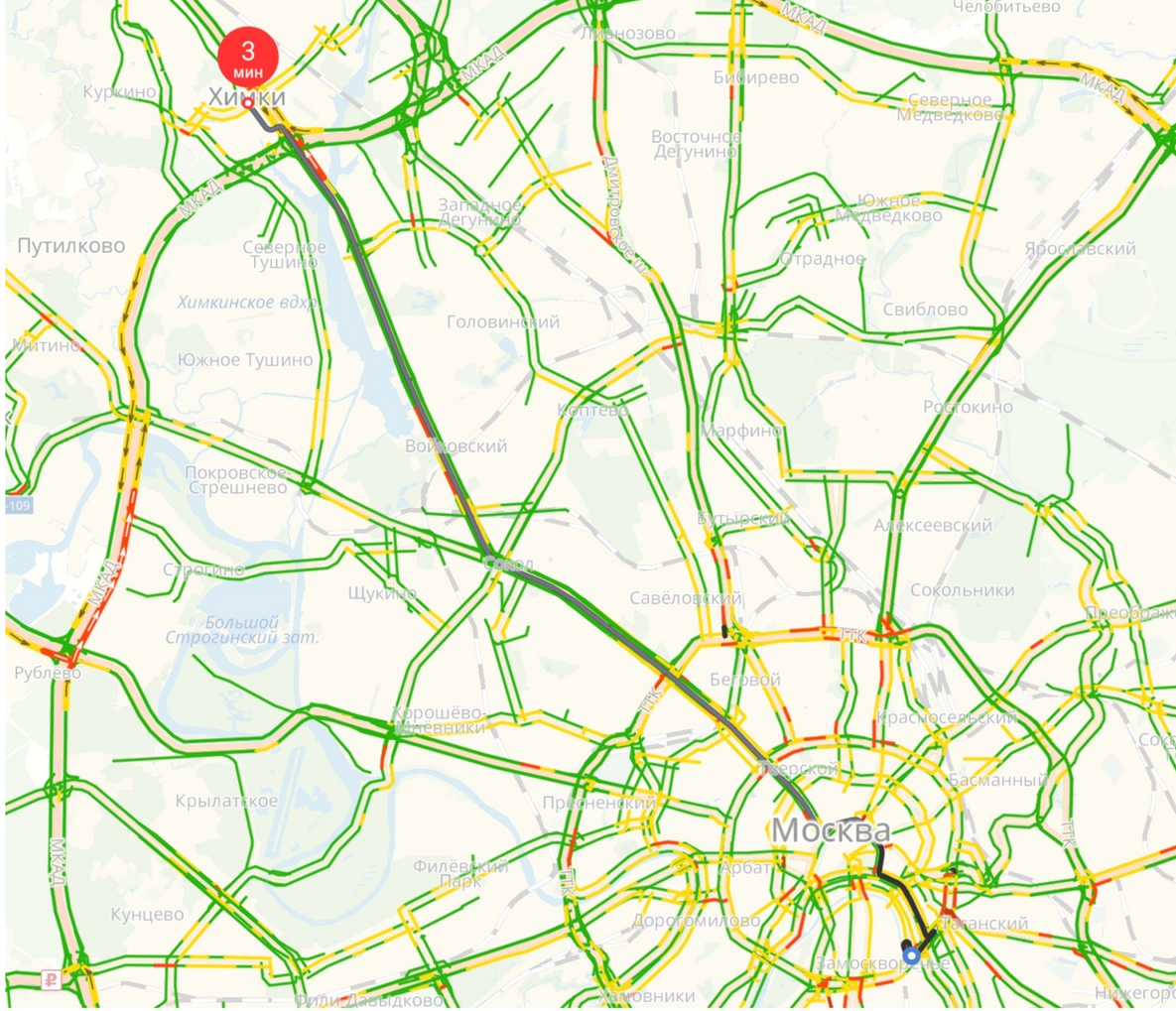
Полухин Антон

Antony Polukhin

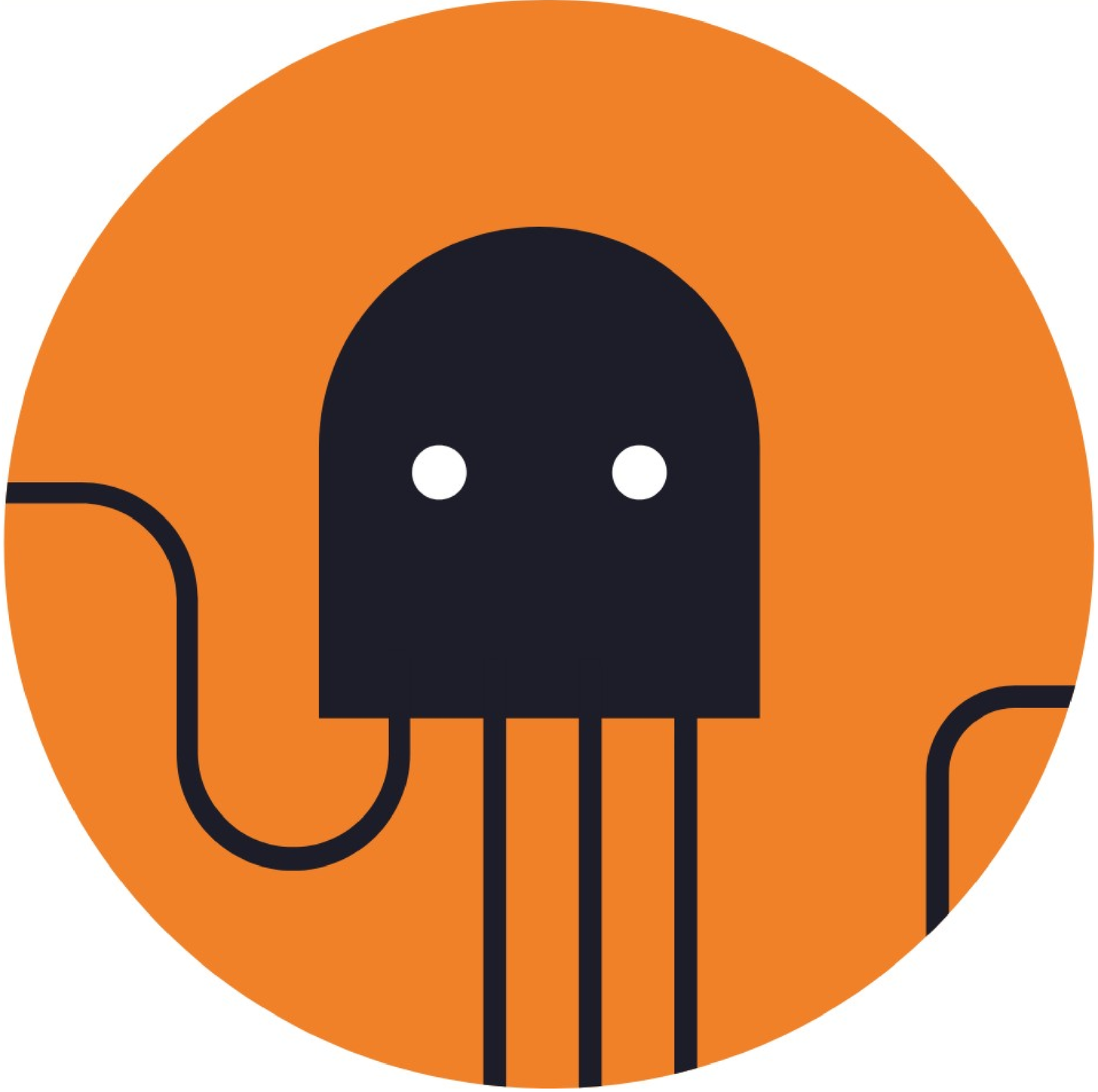
Яндекс Такси

Содержание

- Комментарии — зло
- ...если они не doxygen
- SFINAE — тоже зло



Прочти меня!



Про код

Прочти меня!

Bad code

```
int get(int, unsigned);
std::pair<int, std::pair<int, std::string>> process(bool, int, bool, bool);

std::string sample(int d, std::string (*cb)(int, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != -13) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(result, true, false, true);

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code

```
int get(int, unsigned);
std::pair<int, std::pair<int, std::string>> process(bool, int, bool, bool);

std::string sample(int d, std::string (*cb)(int, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != -13) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(result, true, false, true);

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code

```
int get(int, unsigned);
std::pair<int, std::pair<int, std::string>> process(bool, int, bool, bool);

std::string sample(int d, std::string (*cb)(int, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != -13) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(result, true, false, true);

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(result, true, false, true);

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(result, true, false, true);  <== ERROR

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);  <== ERROR
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, true, false, true);  <== ERROR

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result false, true);

    // Passing processing result to the callback
    return cb(data.second.first, data.second.second);   <== ERROR
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Bad code (Fix #1)

```
enum class Descriptor : int {};
int get(Descriptor, unsigned);
std::pair<Descriptor, std::pair<int, std::string>> process(bool, Descriptor, bool, bool);

std::string sample(Descriptor d, std::string (*cb)(Descriptor, std::string)) {
  // Getting new descriptors from d with a timeout
  auto result = get(d, 1000);

  if (result != Descriptor(-13)) {
    // Descriptor is fine, processing with non bulk options
    auto data = process(true, result, false, true);

    // Passing processing result to the callback
    return cb(data.first, data.second.second);
  }

  // Notifying callback on error
  return cb(result, {});
}
```

Проблемы

Проблемы

- Код написан на **двух** языках

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков
- Комментариев всё ещё недостаточно

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков
- Комментариев всё ещё недостаточно
  - Приходится читать код смежных функций

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков
- Комментариев всё ещё недостаточно
  - **Приходится читать код смежных функций**

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков
- Комментариев всё ещё недостаточно
  - Приходится читать код смежных функций
  - Магические константы

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков
- Комментариев всё ещё недостаточно
  - Приходится читать код смежных функций
  - Магические константы
- Код обработки ошибок и основной логики перемешаны

Проблемы

- Код написан на **двух** языках
  - Кода в два раза больше
  - При отладке проблемы со сверкой 2х языков
- Комментариев всё ещё недостаточно
  - Приходится читать код смежных функций
  - Магические константы
- Код обработки ошибок и основной логики перемешаны
  - Большие блоки кода с большими отступами

Переделаем!

… прошло время

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Good code (Fix #n)

```
std::string Sample(Descriptor listener, std::string (*cb)(Descriptor, std::string)) {
  UASSERT_MSG(cb, "Callback must be a non zero function pointer");

  const auto new_descriptor = Accept(listener, kAcceptTimeout);
  if (new_descriptor == kBadDescriptor) {
    return cb(kBadDescriptor, {});
  }

  auto data = Process(Options::kSync, new_descriptor);
  return cb(data.descriptor, data.payload);
}
```

Приёмы

Приёмы

- Отдельные типы данных

Приёмы

- Отдельные типы данных
  - `void (*accept)(int , int);`

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

- Именованные константы вместо цифр

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

- Именованные константы вместо цифр

```
- Compute(payload, 1023);
```

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

- Именованные константы вместо цифр

```
- Compute(payload, 1023);
+ Compute(payload, kTimeout);
```

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

- Именованные константы вместо цифр

```
- Compute(payload, 1023);
+ Compute(payload, kTimeout);
```

- Enum class вместо bool

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

- Именованные константы вместо цифр

```
- Compute(payload, 1023);
+ Compute(payload, kTimeout);
```

- Enum class вместо bool

```
- Compute(payload, true);
```

Приёмы

- Отдельные типы данных

```
- void (*accept)(int , int);
+ void (*accept)(Descriptor, std::chrono::milliseconds);
```

- Именованные константы вместо цифр

```
- Compute(payload, 1023);
+ Compute(payload, kTimeout);
```

- Enum class вместо bool

```
- Compute(payload, true);
+ Compute(payload, Device::kGPU);
```

Приёмы (часть 2)

Приёмы (часть 2)

- Tuple/Pair не надо использовать

Приёмы (часть 2)

- Tuple/Pair не надо использовать
  - `Compute(data.second.first, data.second.second);`

Приёмы (часть 2)

- Tuple/Pair не надо использовать

```
- Compute(data.second.first, data.second.second);
+ Compute(data.node_id, data.chunk_id);
```

Приёмы (часть 2)

- Tuple/Pair не надо использовать

```
- Compute(data.second.first, data.second.second);
+ Compute(data.node_id, data.chunk_id);

- std::tuple<int, std::string> Receive();
```

Приёмы (часть 2)

- Tuple/Pair не надо использовать

```
- Compute(data.second.first, data.second.second);
+ Compute(data.node_id, data.chunk_id);


- std::tuple<int, std::string> Receive();
+ struct Response {
    int pending_bytes;
    std::string payload;
  };
  Response Receive();
```

Приёмы (часть 3)

Приёмы (часть 3)

- Имена шаблонов

Приёмы (часть 3)

- Имена шаблонов
  - `template <class T> const T& Get();`

Приёмы (часть 3)

- Имена шаблонов

```
- template <class T> const T& Get();
+ template <class Config> const Config& Get();
```

Приёмы (часть 3)

- Имена шаблонов

```
- template <class T> const T& Get();
+ template <class Config> const Config& Get();
```

- Особый namespace для служебных вещей

Приёмы (часть 3)

- Имена шаблонов
  - `- template <class T> const T& Get();`
  - `+ template <class Config> const Config& Get();`
- Особый namespace для служебных вещей
  - `- void Compute(Data data);`

Приёмы (часть 3)

- Имена шаблонов

```
- template <class T> const T& Get();
+ template <class Config> const Config& Get();
```

- Особый namespace для служебных вещей

```
- void Compute(Data data);
+ namespace detail { void Compute(Data data); }
```

Приёмы (часть 3)

- Имена шаблонов

```
- template <class T> const T& Get();
+ template <class Config> const Config& Get();
```

- Особый namespace для служебных вещей

```
- void Compute(Data data);
+ namespace detail { void Compute(Data data); }
```

- Хорошие имена переменных и функций

Приёмы (часть 3)

- Имена шаблонов

```
- template <class T> const T& Get();
+ template <class Config> const Config& Get();
```

- Особый namespace для служебных вещей

```
- void Compute(Data data);
+ namespace detail { void Compute(Data data); }
```

- Хорошие имена переменных и функций

```
- d, cb, mut, Get
```

Приёмы (часть 3)

- Имена шаблонов

```
- template <class T> const T& Get();
+ template <class Config> const Config& Get();
```

- Особый namespace для служебных вещей

```
- void Compute(Data data);
+ namespace detail { void Compute(Data data); }
```

- Хорошие имена переменных и функций

```
- d, cb, mut, Get
+ descriptor, callback, mutator, GetDestination
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

```
- for (const std::pair<int, std::string>& c: map_)
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

```
- for (const std::pair<int, std::string>& c: map_)
+ for (const auto& c: id_name_map_)
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

```
- for (const std::pair<int, std::string>& c: map_)
+ for (const auto& c: id_name_map_)
++ for (const auto& [person_id, first_name]: id_name_map_)
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

```
- for (const std::pair<int, std::string>& c: map_)
+ for (const auto& c: id_name_map_)
++ for (const auto& [person_id, first_name]: id_name_map_)
```

- Не пишите макросы

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

```
- for (const std::pair<int, std::string>& c: map_)
+ for (const auto& c: id_name_map_)
++ for (const auto& [person_id, first_name]: id_name_map_)
```

- Не пишите макросы

```
- X_MACRO(ASD() - 1);
```

Приёмы (часть 4)

- Разный стиль для переменные класса, аргументов функций и констант

```
- connection = address.Connect(timeout);
+ connection_ = address.Connect(kTimeout);
```

- Auto

```
- for (const std::pair<int, std::string>& c: map_)
+ for (const auto& c: id_name_map_)
++ for (const auto& [person_id, first_name]: id_name_map_)
```

- Не пишите макросы

```
- X_MACRO(ASD() - 1);
+ EXPECT_EQ(address.Connect(addr, port, kSockFlag), -1);
```

Приёмы (часть 5)

- Assert ваш друг

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
```

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
+ ASSERT(attempts > 0);
  connection_ = address.Connect(timeout / attempts);
```

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
+ ASSERT(attempts > 0);
  connection_ = address.Connect(timeout / attempts);
```

- Вменяемая вложенность конструкций

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
+ ASSERT(attempts > 0);
  connection_ = address.Connect(timeout / attempts);
```

- Вменяемая вложенность конструкций
- Понятное владение ресурсами

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
+ ASSERT(attempts > 0);
  connection_ = address.Connect(timeout / attempts);
```

- Вменяемая вложенность конструкций

- Понятное владение ресурсами

```
- connection* Connect();
```

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
+ ASSERT(attempts > 0);
  connection_ = address.Connect(timeout / attempts);
```

- Вменяемая вложенность конструкций

- Понятное владение ресурсами

```
- connection* Connect();
+ std::unique_ptr<connection> Connect();
```

Приёмы (часть 5)

- Assert ваш друг

```
- connection_ = address.Connect(timeout / attempts);
+ ASSERT(attempts > 0);
  connection_ = address.Connect(timeout / attempts);
```

- Вменяемая вложенность конструкций

- Понятное владение ресурсами

```
- connection* Connect();
+ std::unique_ptr<connection> Connect();
+ connection& Connect();
```

Приёмы (главная часть)

- Если хочется поставить комментарий

Приёмы (главная часть)

- Если хочется поставить комментарий — попробуйте немного переделать код, чтобы не хотелось

Документация

Прочти меня!

Кто как читает документацию?

Кто как читает документацию?

- Вы тут впервой

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением
- Только знакомитесь с технологией

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением
- Только знакомитесь с технологией:
  - Статьи с описанием подходов и мотивацией

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением
- Только знакомитесь с технологией:
  - Статьи с описанием подходов и мотивацией
- Готовы начать писать код

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением
- Только знакомитесь с технологией:
  - Статьи с описанием подходов и мотивацией
- Готовы начать писать код:
  - Примеры для копирования и модификации

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением
- Только знакомитесь с технологией:
  - Статьи с описанием подходов и мотивацией
- Готовы начать писать код:
  - Примеры для копирования и модификации
- Давно работаете с технологией

Кто как читает документацию?

- Вы тут впервой:
  - Readme.md с ссылками и введением
- Только знакомитесь с технологией:
  - Статьи с описанием подходов и мотивацией
- Готовы начать писать код:
  - Примеры для копирования и модификации
- Давно работаете с технологией:
  - Reference / Index

Что документировать, если имена понятные?

Что документировать, ~~если~~ имена понятные?

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety
- Конфигурирование

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety
- Конфигурирование
  - Файлы конфигурации

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety
- Конфигурирование
  - Файлы конфигурации
  - Переменные окружения

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety
- Конфигурирование
  - Файлы конфигурации
  - Переменные окружения
- Алгоритмические сложности

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety
- Конфигурирование
  - Файлы конфигурации
  - Переменные окружения
- Алгоритмические сложности
- Точки кастомизации

Что документировать, ~~если~~ имена понятные?

То, что невозможно выразить в коде!

- Гарантии
  - Инвалидация итераторов/указателей
  - Signal safety
  - Thread safety
- Конфигурирование
  - Файлы конфигурации
  - Переменные окружения
- Алгоритмические сложности
- Точки кастомизации
- Примеры использования

Хороший пример — работающий пример

Тесты-примеры

Тесты-примеры

- Пишем пример

Тесты-примеры

- Пишем пример
- Добавляем его в тесты

Тесты-примеры

- Пишем пример
- Добавляем его в тесты
- Ссылаемся на него через @snippet

Тесты-примеры

- Пишем пример
- Добавляем его в тесты
- Ссылаемся на него через @snippet
- ???

Тесты-примеры

- Пишем пример
- Добавляем его в тесты
- Ссылаемся на него через @snippet
- ???
- Пример в документации, который всегда актуален и автоматически обновляется

Тесты-примеры

```
/// ## Available options:
///
/// Name | Description | Default value
/// ---- | ----------- | -------------
/// limited-logging-enable | set to true to make LOG_LIMITED drop repeated logs | -
/// limited-logging-interval | utils::StringToDuration suitable duration string to group
repeated logs into one message | -
///
/// ## Config example:
///
/// @snippet components/manager_config_test.cpp Sample logging configurator
```

Тесты-примеры

```
/// ## Available options:
///
/// Name | Description | Default value
/// ---- | ----------- | -------------
/// limited-logging-enable | set to true to make LOG_LIMITED drop repeated logs | -
/// limited-logging-interval | utils::StringToDuration suitable duration string to group
repeated logs into one message | -
///
/// ## Config example:
///
/// @snippet components/manager_config_test.cpp Sample logging configurator
```

Тесты-примеры

```
/// ## Available options:
///
/// Name | Description | Default value
/// ---- | ----------- | -------------
/// limited-logging-enable | set to true to make LOG_LIMITED drop repeated logs | -
/// limited-logging-interval | utils::StringToDuration suitable duration string to group
repeated logs into one message | -
///
/// ## Config example:
///
/// @snippet components/manager_config_test.cpp Sample logging configurator
```

Тесты-примеры

```
/// ## Available options:
///
/// Name | Description | Default value
/// ---- | ----------- | -------------
/// limited-logging-enable | set to true to make LOG_LIMITED drop repeated logs | -
/// limited-logging-interval | utils::StringToDuration suitable duration string to group
repeated logs into one message | -
///
/// ## Config example:
///
/// @snippet components/manager_config_test.cpp Sample logging configurator
```

Тесты-примеры

```
/// ## Available options:
///
/// Name | Description | Default value
/// ---- | ----------- | -------------
/// limited-logging-enable | set to true to make LOG_LIMITED drop repeated logs | -
/// limited-logging-interval | utils::StringToDuration suitable duration string to group
repeated logs into one message | -
///
/// ## Config example:
///
/// @snippet components/manager_config_test.cpp Sample logging configurator
```

Тесты-примеры

```
constexpr char kConfig[] = R"(
components_manager:
  coro_pool:

.......

      task_processor: bg-task-processor

# /// [Sample logging configurator]
    logging-configurator:
      limited-logging-enable: true
      limited-logging-interval: 1s
# /// [Sample logging configurator]
)";
```

Тесты-примеры

```
constexpr char kConfig[] = R"(
components_manager:
  coro_pool:

.......

      task_processor: bg-task-processor

# /// [Sample logging configurator]
    logging-configurator:
      limited-logging-enable: true
      limited-logging-interval: 1s
# /// [Sample logging configurator]
)";
```

Available options:

Name	Description	Default value
limited-logging-enable	set to true to make LOG_LIMITED drop repeated logs	-
limited-logging-interval	**utils::StringToDuration** suitable duration string to group repeated logs into one message	-

Converts strings like "10s", "5d", "1h" to durations.

Config example:

```
logging-configurator:
  limited-logging-enable: true
  limited-logging-interval: 1s
```

C++

Прочти меня!

Не все вещи в C++ диагностируемые

Не все вещи в C++ диагностируемые

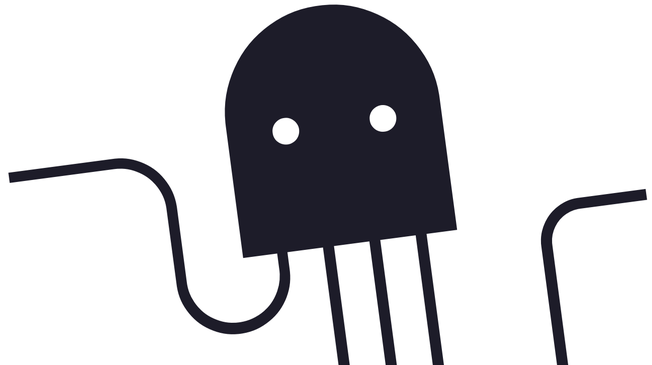
```
<source>:6:5: error: no matching function for call to object of type 'const
__cust_access::_Begin'
    std::ranges::begin(a);
    ^~~~~~~~~~~~~~~~~
/include/c++/11.0.0/bits/ranges_base.h:117:2: note: candidate template ignored:
constraints not satisfied [with _Tp = foo &]
        operator()(_Tp&& __t) const noexcept(_S_noexcept<_Tp>())
        ^
/include/c++/11.0.0/bits/ranges_base.h:114:11: note: because
'is_array_v<remove_reference_t<foo &> >' evaluated to false
        requires is_array_v<remove_reference_t<_Tp>> || __member_begin<_Tp>
                 ^
/include/c++/11.0.0/bits/ranges_base.h:114:50: note: and 'foo &' does not satisfy
'__member_begin'
        requires is_array_v<remove_reference_t<_Tp>> || __member_begin<_Tp>
                                                        ^
```

Не все вещи в C++ диагностируемые

```
/include/c++/11.0.0/bits/iterator_concepts.h:939:33: note: because
'__detail::__decay_copy(__t.begin())' would be invalid: no member named 'begin' in 'foo'
         { __detail::__decay_copy(__t.begin()) } -> input_or_output_iterator;
                                  ^
/include/c++/11.0.0/bits/ranges_base.h:115:7: note: and 'foo &' does not satisfy
'__adl_begin'
         || __adl_begin<_Tp>
            ^
/include/c++/11.0.0/bits/iterator_concepts.h:949:29: note: because
'__detail::__decay_copy(begin(__t))' would be invalid: call to deleted function 'begin'
         { __detail::__decay_copy(begin(__t)) } -> input_or_output_iterator;
                                 ^
1 error generated.
```

Не все вещи в C++ диагностируемые

```
/include/c++/11.0.0/bits/iterator_concepts.h:939:33: note: because
'__detail::__decay_copy(__t.begin())' would be invalid: no member named 'begin' in 'foo'
         { __detail::__decay_copy(__t.begin()) } -> input_or_output_iterator;
                                  ^
/include/c++/11.0.0/bits/ranges_base.h:115:7: note: and 'foo &' does not satisfy
'__adl_begin'
         || __adl_begin<_Tp>
            ^
/include/c++/11.0.0/bits/iterator_concepts.h:949:29: note: because
'__detail::__decay_copy(begin(__t))' would be invalid: call to deleted function 'begin'
         { __detail::__decay_copy(begin(__t)) } -> input_or_output_iterator;
                                  ^
1 error generated.
```



Проблемные техники

- SFINAE
- = delete;

Good SFINAE

```
template <typename T>
T Value::As() const {

  static_assert(formats::common::kHasParseTo<Value, T>,
                "There is no `Parse(const Value&, formats::parse::To<T>)` "
                "in namespace of `T` or `formats::parse`. "
                "Probably you forgot to include the "
                "<formats/parse/common_containers.hpp> or you "
                "have not provided a `Parse` function overload.");

  return Parse(*this, formats::parse::To<T>{});
}
```

Good SFINAE

```
template <typename T>
T Value::As() const {

  static_assert(formats::common::kHasParseTo<Value, T>,
                "There is no `Parse(const Value&, formats::parse::To<T>)` "
                "in namespace of `T` or `formats::parse`. "
                "Probably you forgot to include the "
                "<formats/parse/common_containers.hpp> or you "
                "have not provided a `Parse` function overload.");

  return Parse(*this, formats::parse::To<T>{});
}
```

Good SFINAE

```
template <typename T>
T Value::As() const {

  static_assert(formats::common::kHasParseTo<Value, T>,
                "There is no `Parse(const Value&, formats::parse::To<T>)` "
                "in namespace of `T` or `formats::parse`. "
                "Probably you forgot to include the "
                "<formats/parse/common_containers.hpp> or you "
                "have not provided a `Parse` function overload.");

  return Parse(*this, formats::parse::To<T>{});
}
```

Good SFINAE

```
template <typename T>
T Value::As() const {

  static_assert(formats::common::kHasParseTo<Value, T>,
                "There is no `Parse(const Value&, formats::parse::To<T>)` "
                "in namespace of `T` or `formats::parse`. "
                "Probably you forgot to include the "
                "<formats/parse/common_containers.hpp> or you "
                "have not provided a `Parse` function overload.");

  return Parse(*this, formats::parse::To<T>{});
}
```

Good SFINAE

```
template <typename T>
T Value::As() const {

  static_assert(formats::common::kHasParseTo<Value, T>,
                "There is no `Parse(const Value&, formats::parse::To<T>)` "
                "in namespace of `T` or `formats::parse`. "
                "Probably you forgot to include the "
                "<formats/parse/common_containers.hpp> or you "
                "have not provided a `Parse` function overload.");

  return Parse(*this, formats::parse::To<T>{});
}
```

Good SFINAE

```
template <typename T>
T Value::As() const {

  static_assert(formats::common::kHasParseTo<Value, T>,
                "There is no `Parse(const Value&, formats::parse::To<T>)` "
                "in namespace of `T` or `formats::parse`. "
                "Probably you forgot to include the "
                "<formats/parse/common_containers.hpp> or you "
                "have not provided a `Parse` function overload.");

  return Parse(*this, formats::parse::To<T>{});
}
```

Good SFINAE

```
static_assert
```

Спасибо

Полухин Антон

Эксперт-разработчик C++

antoshkka@gmail.com

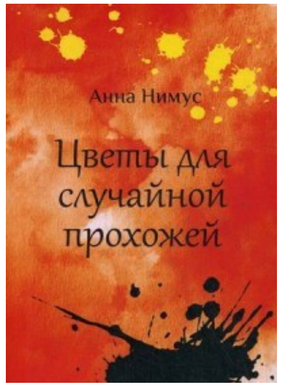
antoshkka@yandex-team.ru

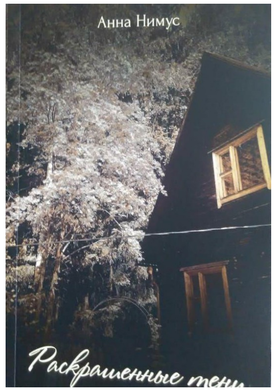
https://github.com/apolukhin

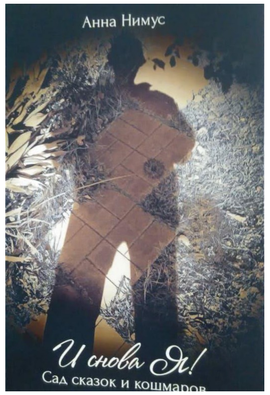
C++ РГ21 C++ РОССИЯ https://stdcpp.ru/

https://t.me/CppQuizzBot

Спасибо


Анна Нимус
Цветы для случайной прохожей


Анна Нимус


Анна Нимус
И снова Я!
Сад сказок и кошмаров