Киберпанк C++77

Применение рефлексии на коленке с конкретными примерами и сценариями

Спикер: Артем Бабаев
Должность: Руководитель группы
Компания: Redkit Lab
Почта: ivavpi@gmail.com
tg: @simplepersonru1
vk: id752278665

г. Москва, 2026 г.

План

1. Мотивация
2. Короткая затравочка
3. Аналоги
4. Применение статической рефлексии
5. Практическая польза
6. Внутреннее устройство
7. Общеприменительные полезные артефакты
8. Ссылка на репозиторий

Затравочка

```cpp
struct Employee
{
   std::string name;
   int age = {};
   bool onVacation = false;
   int experience = {};
};
```

Затравочка

```
/// @Json
struct Employee
{
   std::string name;
   int age = {};
   /// @Json{ key: on_vacation }
   bool onVacation = false;
   /// @Json{ ignore: true }
   int experience = {};
};
```

Затравочка

```cpp
/// @Json
struct Employee
{
    std::string name;
    int age = {};
    /// @Json{ key: on_vacation }
    bool onVacation = false;
    /// @Json{ ignore: true }
    int experience = {};
};
```

```cpp
int main()
{
    auto bob = Employee {
        .name = "Bob",
        .age = 30,
        .onVacation = true,
        .experience = 7
    };

    std::string json = rgen::toJson<Employee>(bob);
    std::println("{}", json);
    // {"age":30,"name":"Bob","on_vacation":true}
}
```

Затравочка

```cpp
/// @Json
struct Employee
{
    std::string name;
    int age = {};
    /// @Json{ key: on_vacation }
    bool onVacation = false;
    /// @Json{ ignore: true }
    int experience = {};
};
```

```cpp
int main()
{
    auto bob = Employee {
        .name = "Bob",
        .age = 30,
        .onVacation = true,
        .experience = 7
    };

    std::string json = rgen::toJson<Employee>(bob);
    std::println("{}", json);
    // {"age":30,"name":"Bob","on_vacation":true}
}
```

Затравочка

```
std::string json = rgen::toJson<Employee>(bob);
```

Затравочка

```
std::string json = rgen::toJson<Employee>(bob);

template <typename T>
std::string toJson(const T& obj)
{
    nlohmann::json j;
    fromTypeToJson(j, obj);
    return j.dump();
}
```

Затравочка

```
std::string json = rgen::toJson<Employee>(bob);

template <typename T>
std::string toJson(const T& obj)
{
    nlohmann::json j;
    fromTypeToJson(j, obj);
    return j.dump();
}
```

Затравочка

```
template <typename T>
void fromTypeToJson(nlohmann::json& json, const T& val);
```

Сгенерировано:

```
template<>
void fromTypeToJson(nlohmann::json& json, const Employee& val)
{
   fromTypeToJson(json["name"], val.name);
   fromTypeToJson(json["age"], val.age);
   fromTypeToJson(json["on_vacation"], val.onVacation);
}
```

Затравочка

```
template <typename T>
void fromTypeToJson(nlohmann::json& json, const T& val)
requires std::is_integral_v<T> || std::is_floating_point_v<T>
{
   json = val;
}

template <>
inline void fromTypeToJson(nlohmann::json& json, const std::string& val)
{
   json = val;
}

// ...
```

Затравочка

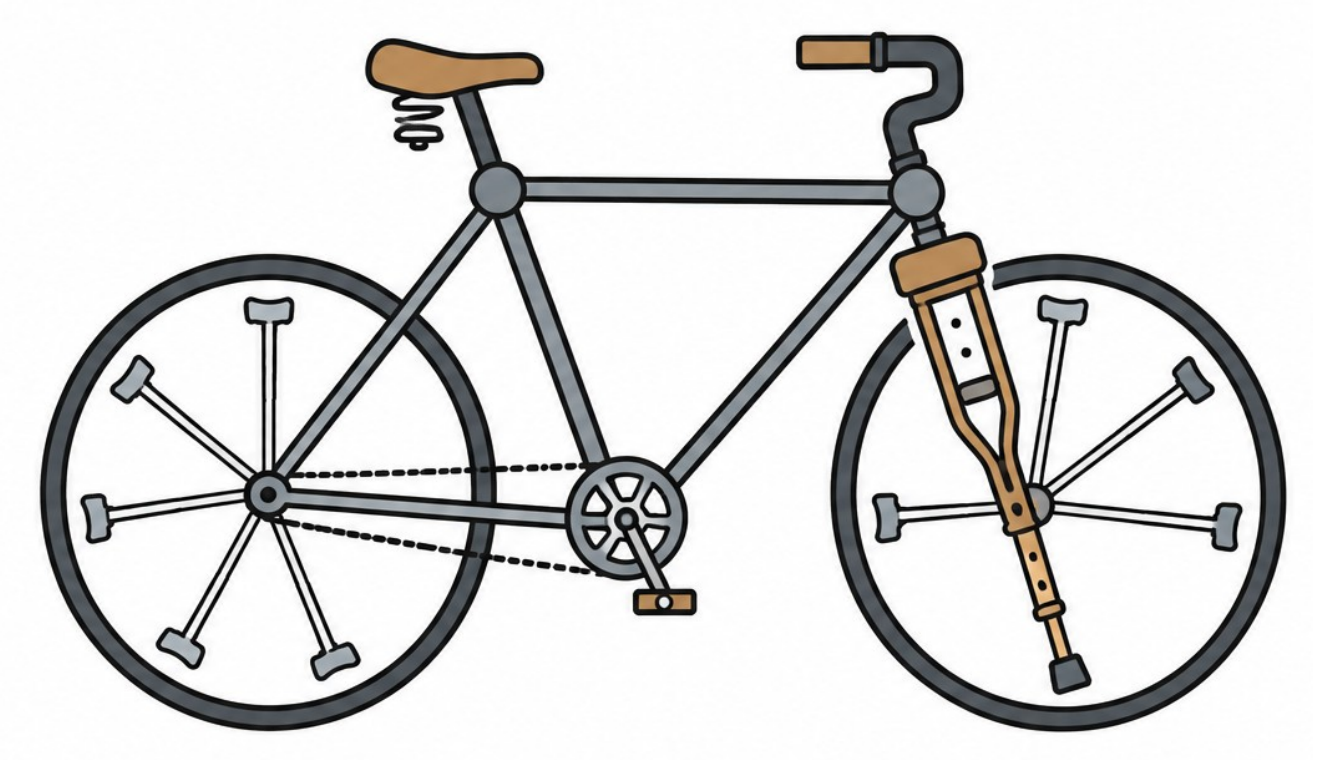
```cpp
/// @Json
struct Employee
{
    std::string name;
    int age = {};
    /// @Json{ key: on_vacation }
    bool onVacation = false;
    /// @Json{ ignore: true }
    int experience = {};
};
```

Java

```java
public class Employee
{
    public String name;
    public int age;

    @JsonProperty("on_vacation")
    public boolean onVacation;

    @JsonIgnore
    public int experience;
}
```

author: Jeremy Kratz, source: https://github.com/isocpp/logos, license: isocpp.org

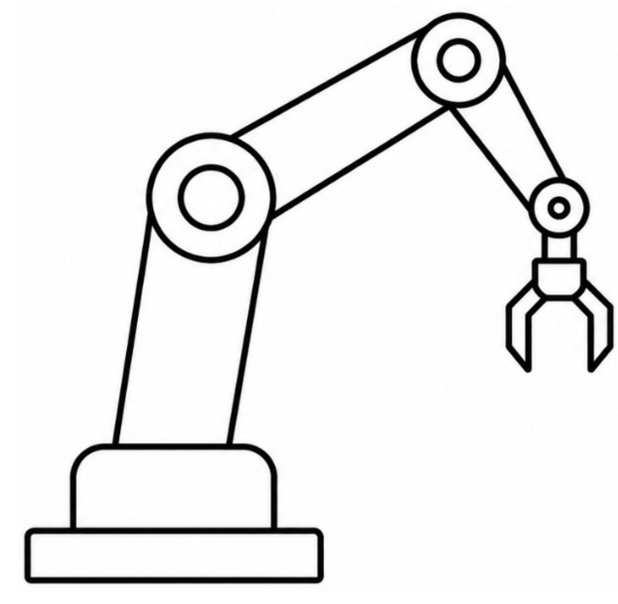
Мотивация



Изображение сгенерировано https://chatgpt.com/

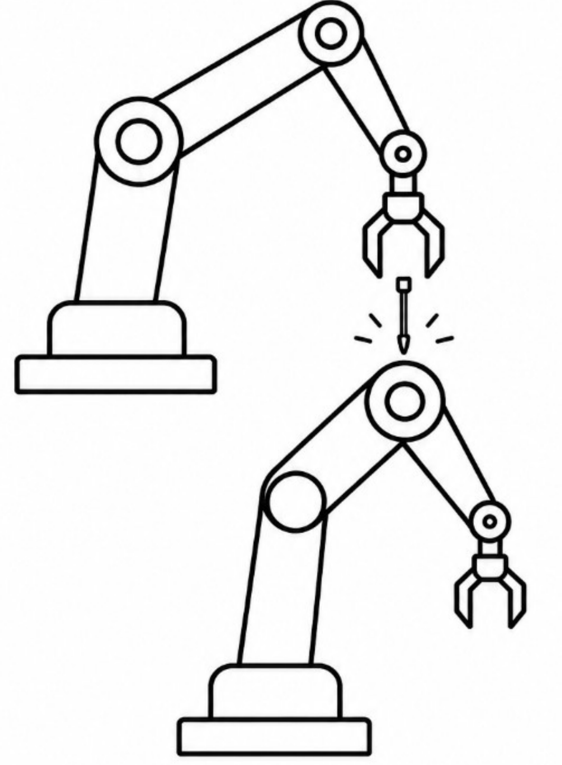
Мотивация

Автоматизация



Изображение сгенерировано https://chatgpt.com/

Автоматизация
с рефлексией



Изображение сгенерировано https://chatgpt.com/

Наткнулся на статью

Евгений Шульгин
Кодогенератор Waffle++ для C++

https://habr.com/ru/articles/710744/

Аналоги в C++

1. C++26
   Очень ждем

Аналоги в C++

1. C++26
   Очень ждем

2. Boost.PFR, Magic Enum и подобные
   Выжимают все что можно из шаблонной магии, очень просто использовать, но не все сценарии применения возможны

Аналоги в C++

1. C++26
   Очень ждем

2. Boost.PFR, Magic Enum и подобные
   Выжимают все что можно из шаблонной магии, очень просто использовать, но не все сценарии применения возможны

3. easy_reflection_cpp, clReflect, …
   Опираются на frontend компилятора (плагин к компилятору, анализ через clangAST, мэтчеры и т.д.)

Дождались примеров

Изображение сгенерировано https://chatgpt.com/

Derive

```
use serde::{Serialize, Deserialize};

#[derive(Clone, PartialEq,
         Serialize, Deserialize)]
struct User {
   name: String,
   age: u32,
}
```

Derive

```rust
use serde::{Serialize, Deserialize};

#[derive(Clone, PartialEq,
         Serialize, Deserialize)]
struct User {
   name: String,
   age: u32,
}
```

Derive

```rust
use serde::{Serialize, Deserialize};

#[derive(Clone, PartialEq,
         Serialize, Deserialize)]
struct User {
    name: String,
    age: u32,
}
```

```rust
fn main() {
    let alice = User {
        name: "Alice".to_string(),
        age: 30,
    };

    let cloned = alice.clone();
    println!("Equal? {}", alice == cloned); // true

    let json = serde_json::to_string(&alice).unwrap();
    println!("{}", json); // { "name": "Alice", "age": 30 }
}
```

Derive

```rust
use serde::{Serialize, Deserialize};

#[derive(Clone, PartialEq,
         Serialize, Deserialize)]
struct User {
    name: String,
    age: u32,
}
```

```rust
fn main() {
    let alice = User {
        name: "Alice".to_string(),
        age: 30,
    };

    let cloned = alice.clone();
    println!("Equal? {}", alice == cloned); // true

    let json = serde_json::to_string(&alice).unwrap();
    println!("{}", json); // { "name": "Alice", "age": 30 }
}
```

Derive

```
#include <rgen/derive.h>
#include <string>

/// @Derive
struct User
{
   RGEN_DERIVE(friend, User,
               RGEN_Equals, RGEN_Json)

   std::string name;
   int age = {};
};
```

Derive

```cpp
#include <rgen/derive.h>
#include <string>

/// @Derive
struct User
{
   RGEN_DERIVE(friend, User,
               RGEN_Equals, RGEN_Json)

   std::string name;
   int age = {};
};
```

```cpp
int main()
{
   auto alice = User {
       .name = "Alice",
       .age = 30
   };

   auto copy = alice;
   std::println("{}", copy == alice);
   // true

   auto json = rgen::toJson<User>(alice);
   std::println("{}", json);
   // {"age":30,"name":"Alice"}
}
```

Derive

```
#define INTERNAL_DERIVE_1(Prefix, Type, a1) \
    a1(Prefix, Type)
#define INTERNAL_DERIVE_2(Prefix, Type, a1, a2) \
    a1(Prefix, Type) INTERNAL_DERIVE_1(Prefix, Type, a2)
// ... и так 9 раз

// Макрос для выбора нужного количества аргументов
#define INTERNAL_DERIVE_SELECT(_1, _2, N, ...) N
#define RGEN_DERIVE(Prefix, Type, ...) \
    INTERNAL_DERIVE_SELECT(__VA_ARGS__, \
                           INTERNAL_DERIVE_2, \
                           INTERNAL_DERIVE_1) \
    (Prefix, Type, __VA_ARGS__)
```

Derive

```
RGEN_DERIVE(friend, User,
            RGEN_Equals, RGEN_Json)

==

RGEN_Equals(friend, User)
RGEN_Json(friend, User)
```

Derive

```
#define RGEN_Equals(Prefix, Type) \
    Prefix bool operator==(const Type& lhs, const Type& rhs); \
    Prefix bool operator!=(const Type& lhs, const Type& rhs);

#define RGEN_Less(Prefix, Type) \
    Prefix bool operator<(const Type& lhs, const Type& rhs);
```

Derive

```
#define RGEN_Equals(Prefix, Type) \
    Prefix bool operator==(const Type& lhs, const Type& rhs); \
    Prefix bool operator!=(const Type& lhs, const Type& rhs);

#define RGEN_Less(Prefix, Type) \
    Prefix bool operator<(const Type& lhs, const Type& rhs);

#define RGEN_QStream(Prefix, Type)                                   \
    Prefix QDataStream& operator<<(QDataStream& out, const Type& info); \
    Prefix QDataStream& operator>>(QDataStream& in, Type& info);
```

Derive

```
#define RGEN_Equals(Prefix, Type) \
    Prefix bool operator==(const Type& lhs, const Type& rhs); \
    Prefix bool operator!=(const Type& lhs, const Type& rhs);

#define RGEN_Less(Prefix, Type) \
    Prefix bool operator<(const Type& lhs, const Type& rhs);

#define RGEN_QStream(Prefix, Type)                               \
    Prefix QDataStream& operator<<(QDataStream& out, const Type& info); \
    Prefix QDataStream& operator>>(QDataStream& in, Type& info);
```

- RGEN_Json
- RGEN_Hasher (быть ключом std хэш-мапов)
- RGEN_QHasher (быть ключом Qt хэш-мапов)
- RGEN_QVariantMap (сериализация в QVariantMap)
- ...

Derive

```
/// @Derive
struct Point
{
   RGEN_DERIVE(friend, Point,
               RGEN_Equals, RGEN_Less)

   double x = {};
   double y = {};
};
```

```
template <typename T>
struct PointTempl
{
   T x = {};
   T y = {};
}

/// @Derive
using PointInt = PointTempl<int>;
RGEN_DERIVE(, PointInt, RGEN_Equals, RGEN_Less);
```

Derive

```
#ifdef __linux__
#   define DECL_EXPORT     __attribute__((visibility("default")))
#   define DECL_IMPORT     __attribute__((visibility("default")))
#elif defined(_WIN64) || defined(_WIN32)
#   define DECL_EXPORT     __declspec(dllexport)
#   define DECL_IMPORT     __declspec(dllimport)
#else
#   define DECL_EXPORT
#   define DECL_IMPORT
#endif
```

Derive

```cpp
/// @Derive
struct Point
{
   RGEN_DERIVE(friend DECL_EXPORT, Point,
               RGEN_Equals, RGEN_Less)

   double x = {};
   double y = {};
};
```

```cpp
template <typename T>
struct PointTempl
{
   T x = {};
   T y = {};
}

/// @Derive
using PointInt = PointTempl<int>;
RGEN_DERIVE(DECL_EXPORT, PointInt,
            RGEN_Equals, RGEN_Less);
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
#define RGEN_Equals(Prefix, Type) \
    Prefix bool operator==(const Type& lhs, const Type& rhs); \
    Prefix bool operator!=(const Type& lhs, const Type& rhs);
```

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
#define RGEN_Equals(Prefix, Type) \
   Prefix bool operator==(const Type& lhs, const Type& rhs); \
   Prefix bool operator!=(const Type& lhs, const Type& rhs);
```

```
bool operator!=(const Type& lhs, const Type& rhs) {
    return !(lhs == rhs);
}
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
/// @Equals
struct Point
{
    RGEN_Equals(friend, Point)

    double x = {};
    double y = {};
};
```

```
bool operator==(const Point& lhs, const Point& rhs) {
    bool res = true;
    res &= rgen::equals(lhs.x, rhs.x);
    res &= rgen::equals(lhs.y, rhs.y);
    return res;
}
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```cpp
/// @Equals
struct Point
{
    RGEN_Equals(friend, Point)

    double x = {};
    double y = {};
};
```

```cpp
bool operator==(const Point& lhs, const Point& rhs) {
    bool res = true;
    res &= rgen::equals(lhs.x, rhs.x);
    res &= rgen::equals(lhs.y, rhs.y);
    return res;
}
```

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
template <typename T>
bool equals(const T& lhs, const T& rhs) {
   return lhs == rhs;
}

template <>
inline bool equals<double>(const double& lhs, const double& rhs) {
   /// Реализация под числа с плавающей точкой
}

template <>
inline bool equals<float>(const float& lhs, const float& rhs) {
   /// Реализация под числа с плавающей точкой
}
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
#define RGEN_Less(Prefix, Type) \
    Prefix bool operator<(const Type& lhs, const Type& rhs);
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```cpp
/// @Less
struct Date
{
    RGEN_Less(friend, Date)

    int year = {};
    int month = {};
    int day = {};
};
```

```cpp
bool operator<(const Point& lhs, const Point& rhs) {
    return std::make_tuple(
        lhs.year,
        lhs.month,
        lhs.day
    ) < std::make_tuple(
        rhs.year,
        rhs.month,
        rhs.day
    );
}
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
#define RGEN_QStream(Prefix, Type)                                    \
   Prefix QDataStream& operator<<(QDataStream& out, const Type& info); \
   Prefix QDataStream& operator>>(QDataStream& in, Type& info);
```

Redkit Lab

Операторы: ‘==’, ‘<’, ‘<<’, ...

```
/// @Equals
struct Point
{
    RGEN_QStream(friend, Point)

    double x = {};
    double y = {};
};
```

```
QDataStream& operator<<(QDataStream& out,
                        const Point& obj) {
    out
    << obj.x
    << obj.y;
    return out;
}
```

Enum

Источник: общественное достояние

Enum

```
enum class Color
{
   Red,
   Green,
   Blue
}
```

Enum

```
enum class Color
{
    Red,
    Green,
    Blue
}
```

```
void main()
{
    if (foo::toString(Color::Red) == "Red")
        std::println("nice!");
}
```

Enum

```
enum class Color
{
   Red,    ///< Красный
   Green, ///< Зеленый
   Blue   ///< Синий
}
```

Enum

```cpp
/// @Enum
enum class Color
{
    Red,    ///< Красный
    Green, ///< Зеленый
    Blue    ///< Синий
}
```

```cpp
void main()
{
    if (rgen::Enum<Color>::toString(Color::Red)
        == "Красный") {
        std::println("nice!");
    }
}
```

JsonSchema

```cpp
enum class ContactType
{
    Email = 0,
    Telephone = 1
}

/// @JsonSchema
struct User
{
    std::string name;
    int age; ///< Возраст
    ContactType contact;
}
```

```json
{
 "$schema": "https://json-schema...",
 "properties": {
   "age": {
     "description": "Возраст",
     "type": "integer"
   },
   "contact": {
     "enum": [ 0, 1 ],
     "type": "integer"
   },
   "name": { "type": "string" }
 },
 "title": "User",
 "type": "object"
}
```

JsonSchema

```cpp
enum class ContactType
{
    Email = 0,
    Telephone = 1
}

/// @JsonSchema
struct User
{
    std::string name;
    int age; ///< Возраст
    ContactType contact;
}
```

```json
{
 "$schema": "https://json-schema...",
 "properties": {
   "age": {
     "description": "Возраст",
     "type": "integer"
   },
   "contact": {
     "enum": [ 0, 1 ],
     "type": "integer"
   },
   "name": { "type": "string" }
 },
 "title": "User",
 "type": "object"
}
```

JsonSchema

```cpp
enum class ContactType
{
    Email = 0,
    Telephone = 1
}

/// @JsonSchema
struct User
{
    std::string name;
    int age; ///< Возраст
    ContactType contact;
}
```

```json
{
 "$schema": "https://json-schema...",
 "properties": {
   "age": {
     "description": "Возраст",
     "type": "integer"
   },
   "contact": {
     "enum": [ 0, 1 ],
     "type": "integer"
   },
   "name": { "type": "string" }
 },
 "title": "User",
 "type": "object"
}
```

JsonSchema

```cpp
/// @JsonSchema
struct Company
{
   User director;
   bool certified;
   std::vector<User> users;
};
```

```json
{
 "$schema": "https://json-schema...",
 "properties": {
   "certified": { "type": "boolean" },
   "director": {...},
   "users": {
     "items": {
       "$ref": "#/properties/director"
     },
     "type": "array"
   }
 },
 "title": "Company",
 "type": "object"
}
```

JsonSchema

```cpp
/// @JsonSchema
struct Company
{
   User director;
   bool certified;
   std::vector<User> users;
};
```

```json
{
 "$schema": "https://json-schema...",
 "properties": {
   "certified": { "type": "boolean" },
   "director": {...},
   "users": {
     "items": {
       "$ref": "#/properties/director"
     },
     "type": "array"
   }
 },
 "title": "Company",
 "type": "object"
}
```

JsonSchema

```cpp
/// @JsonSchema
struct Company
{
   User director;
   bool certified;
   std::vector<User> users;
};
```

```json
{
 "$schema": "https://json-schema...",
 "properties": {
   "certified": { "type": "boolean" },
   "director": {...},
   "users": {
     "items": {
       "$ref": "#/properties/director"
     },
     "type": "array"
   }
 },
 "title": "Company",
 "type": "object"
}
```

MetaModel

```
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
   /// @Display Имя
   /// @MetaModel { min: 1, max: 100 }
   std::string name;
   /// @Display Возраст
   /// @MetaModel { min: 0, max: 1000 }
   int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
   /// @Display Друзья
   std::vector<std::shared_ptr<Living>> friends;
};
```

Redkit Lab

MetaModel ❤️

```
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
    /// @Display Имя
    /// @MetaModel { min: 1, max: 100 }
    std::string name;
    /// @Display Возраст
    /// @MetaModel { min: 0, max: 1000 }
    int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
    /// @Display Друзья
    std::vector<std::shared_ptr<Living>> friends;
};
```

Redkit Lab

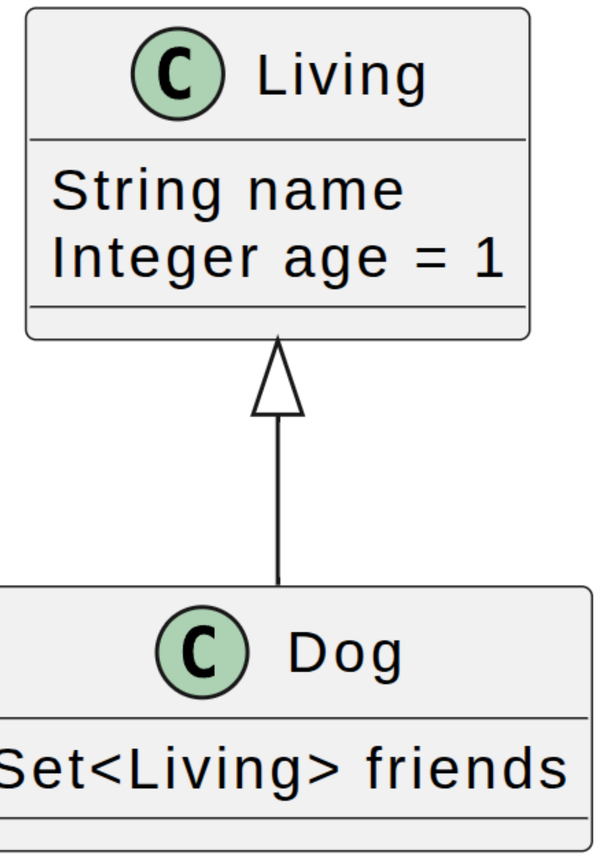
MetaModel ❤️

```
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
   /// @Display Имя
   /// @MetaModel { min: 1, max: 100 }
   std::string name;
   /// @Display Возраст
   /// @MetaModel { min: 0, max: 1000 }
   int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
   /// @Display Друзья
   std::vector<std::shared_ptr<Living>> friends;
};
```

Redkit Lab

MetaModel

```
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
   /// @Display Имя
   /// @MetaModel { min: 1, max: 100 }
   std::string name;
   /// @Display Возраст
   /// @MetaModel { min: 0, max: 1000 }
   int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
   /// @Display Друзья
   std::vector<std::shared_ptr<Living>> friends;
};
```



Redkit Lab

MetaModel

```cpp
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
   /// @Display Имя
   /// @MetaModel { min: 1, max: 100 }
   std::string name;
   /// @Display Возраст
   /// @MetaModel { min: 0, max: 1000 }
   int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
   /// @Display Друзья
   std::vector<std::shared_ptr<Living>> friends;
};
```

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org...">
   <Dog rdf:about="#_db9a4e32">
      <Living.name>Барсук</Living.name>
      <Living.age>3</Living.age>
   </Dog>
   <Dog rdf:about="#_ac9a4e32">
      <Living.name>Петя</Living.name>
      <Living.age>2</Living.age>
      <Dog.friends rdf:resource="#_nf9a4e32"/>
   </Dog>
   <Dog rdf:about="#_nf9a4e32">
      <Living.name>Барбос</Living.name>
      <Living.age>15</Living.age>
      <Dog.friends rdf:resource="#_ac9a4e32"/>
      <Dog.friends rdf:resource="#_db9a4e32"/>
   </Dog> </rdf:RDF>
```

Redkit Lab

MetaModel

```cpp
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
    /// @Display Имя
    /// @MetaModel { min: 1, max: 100 }
    std::string name;
    /// @Display Возраст
    /// @MetaModel { min: 0, max: 1000 }
    int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
    /// @Display Друзья
    std::vector<std::shared_ptr<Living>> friends;
};
```

```sql
CREATE TABLE Living (
-- ...
CONSTRAINT min_age CHECK ("age" >= 0),
CONSTRAINT max_age CHECK ("age" <= 1000),
CONSTRAINT min_name CHECK (LENGTH("name") >= 1),
CONSTRAINT max_name CHECK (LENGTH("name") <= 100),
);
```

author: alaa kaddour, source , license: CC-BY-SA-4.0

MetaModel

```cpp
/// @MetaModel { type: Class }
/// @Display Существо
struct Living : mml::BaseClass {
    /// @Display Имя
    /// @MetaModel { min: 1, max: 100 }
    std::string name;
    /// @Display Возраст
    /// @MetaModel { min: 0, max: 1000 }
    int age = 1;
};
/// @MetaModel { type: Class }
/// @Display Собака
struct Dog : Living {
    /// @Display Друзья
    std::vector<std::shared_ptr<Living>> friends;
};
```

DOG : Buddy

Имя (1–100)

Buddy

Возраст (0–1000)

3 — 3

Друзья Living[ ]

R Rex ×

- [x] R Rex — возраст 4
- [ ] M Max — возраст 2

1 из 2 выбрано

Отмена | Сохранить

макет сгенерирован https://claude.ai/

MetaModel

1 точка истины в коде

На выходе:

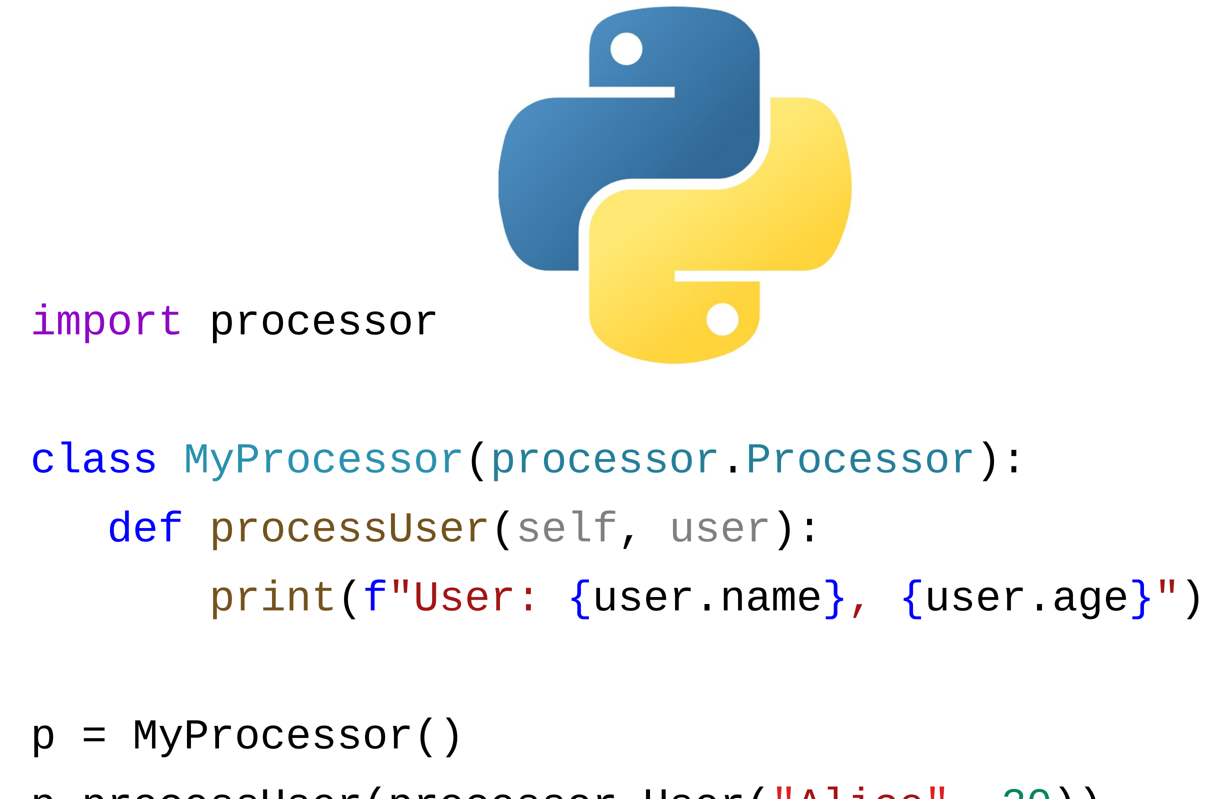
- PlantUml диаграмма
- RDFXML представление
- CONSTRAINTS для SQL БД
- Заготовки для GUI
- Статическая документация.md

Python binding

```
struct User {
    std::string name;
    int age;
};


class Processor {
public:
    virtual void processUser(User user) = 0;
    virtual ~Processor() = default;
};
```

Python binding

```cpp
struct User {
   std::string name;
   int age;
};


class Processor {
public:
   virtual void processUser(User user) = 0;
   virtual ~Processor() = default;
};
```



```python
import processor

class MyProcessor(processor.Processor):
   def processUser(self, user):
       print(f"User: {user.name}, {user.age}")

p = MyProcessor()
p.processUser(processor.User("Alice", 30))
```

author: Tim Parkin, source , license: PSF Trademark Usage Policy

Swig (Python binding)

```cpp
struct User {
    std::string name;
    int age;
};


class Processor {
public:
    virtual void processUser(User user) = 0;
    virtual ~Processor() = default;
};
```

```
%module processor
%{
#include "user.h"
%}
%include "std_string.i"
struct User {
    std::string name;
    int age;
};
%feature("director") Processor;
class Processor {
public:
    virtual void processUser(User user);
    virtual ~Processor();
};
```

PyBind11 (Python binding)

```
class PyProcessor : public Processor {
public:
    using Processor::Processor;

    void processUser(User user) override {
        PYBIND11_OVERRIDE_PURE(
            void,
            Processor,
            processUser,
            user
        );
    }
};
```

```
PYBIND11_MODULE(processor_pb, m) {
    m.doc() = "PyBind11 example";

    py::class_<User>(m, "User")
        .def(py::init<>())
        .def_readwrite("name", &User::name)
        .def_readwrite("age", &User::age)
        .def(py::init<const std::string&, int>());

    py::class_<Processor, PyProcessor>(m, "Processor")
        .def(py::init<>())
        .def("processUser", &Processor::processUser);
}
```

Swig (Python binding)

```cpp
/// @Swig
struct User {
    std::string name;
    int age;
};

/// @Swig
class Processor {
public:
    virtual void processUser(User user) = 0;
    virtual ~Processor() = default;
};
```

```
%module processor
%{
#include "user.h"
%}
%include "std_string.i"
struct User {
    std::string name;
    int age;
};
%feature("director") Processor;
class Processor {
public:
    virtual void processUser(User user);
    virtual ~Processor();
};
```

Интересные примеры использования

```
class Turtle
{
public:
    virtual ~Turtle() = default;
    virtual void PenUp() = 0;
    virtual void PenDown() = 0;
    virtual void Turn(int degrees) = 0;
    virtual int GetX() const = 0;
}
```

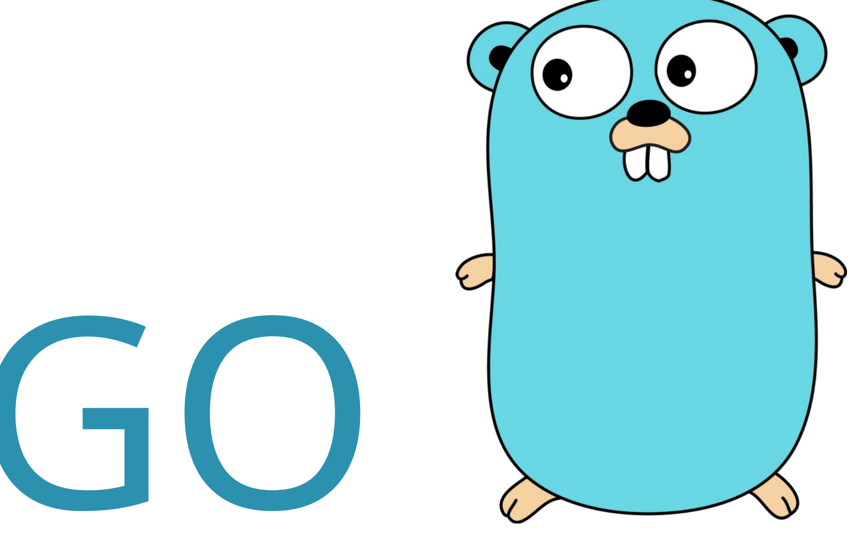
Интересные примеры использования

```cpp
/// @GMock
class Turtle
{
public:
    virtual ~Turtle() = default;
    virtual void PenUp() = 0;
    virtual void PenDown() = 0;
    virtual void Turn(int degrees) = 0;
    virtual int GetX() const = 0;
}
```

```cpp
#include <gmock/gmock.h>
namespace rgen {

class MockTurtle : public Turtle
{
public:
    MOCK_METHOD(void, PenUp, (), (override));
    MOCK_METHOD(void, PenDown, (), (override));
    MOCK_METHOD(void, Turn, (int degrees), (override));
    MOCK_METHOD(int, GetX, (), (const override));
}

} // namespace rgen
```

Интересные примеры использования

```
type Quacker interface {
    Quack() string
}
```

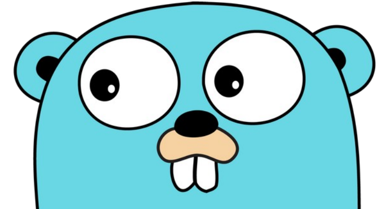


author: Renee French, source, license: CC BY 4.0

Интересные примеры использования

```
type Quacker interface {
    Quack() string
}
```

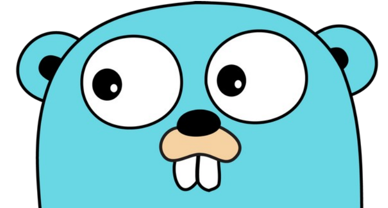
```
type Duck struct{ Name string }
func (d Duck) Quack() string {
    return "утка " + d.Name + " крякает"
}

func MakeItQuack(q Quacker) {
    fmt.Println(q.Quack())
}

func main() {
    duck := Duck{Name: "Дональд"}
    MakeItQuack(duck)
}
```



Интересные примеры использования

```
type Quacker interface {
    Quack() string
}
```

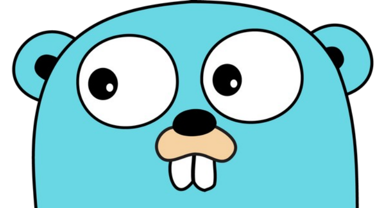
```
type Duck struct{ Name string }
func (d Duck) Quack() string {
    return "утка " + d.Name + " крякает"
}

func MakeItQuack(q Quacker) {
    fmt.Println(q.Quack())
}

func main() {
    duck := Duck{Name: "Дональд"}
    MakeItQuack(duck)
}
```



Интересные примеры использования

```go
type Quacker interface {
    Quack() string
}
```

```go
type Duck struct{ Name string }
func (d Duck) Quack() string {
    return "утка " + d.Name + " крякает"
}

func MakeItQuack(q Quacker) {
    fmt.Println(q.Quack())
}

func main() {
    duck := Duck{Name: "Дональд"}
    MakeItQuack(duck)
}
```



Интересные примеры использования

```
struct IContainer
{
   int size() const;
   bool empty() const;
   void clear();
}
```

Интересные примеры использования

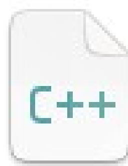
```cpp
/// @polymorphic
struct IContainer
{
    int size() const;
    bool empty() const;
    void clear();
}
```

```cpp
void clearIfNotEmpty(rgen::poly<IContainer> c)
{
    if (c.empty())
        return;
    std::println("old size: {}", c.size());
    c.clear();
}

int main()
{
    clearIfNotEmpty(std::vector<int>{});
    clearIfNotEmpty(std::deque<double>{});
    clearIfNotEmpty(std::map<std::string, int>{});
}
```

Внутреннее устройство

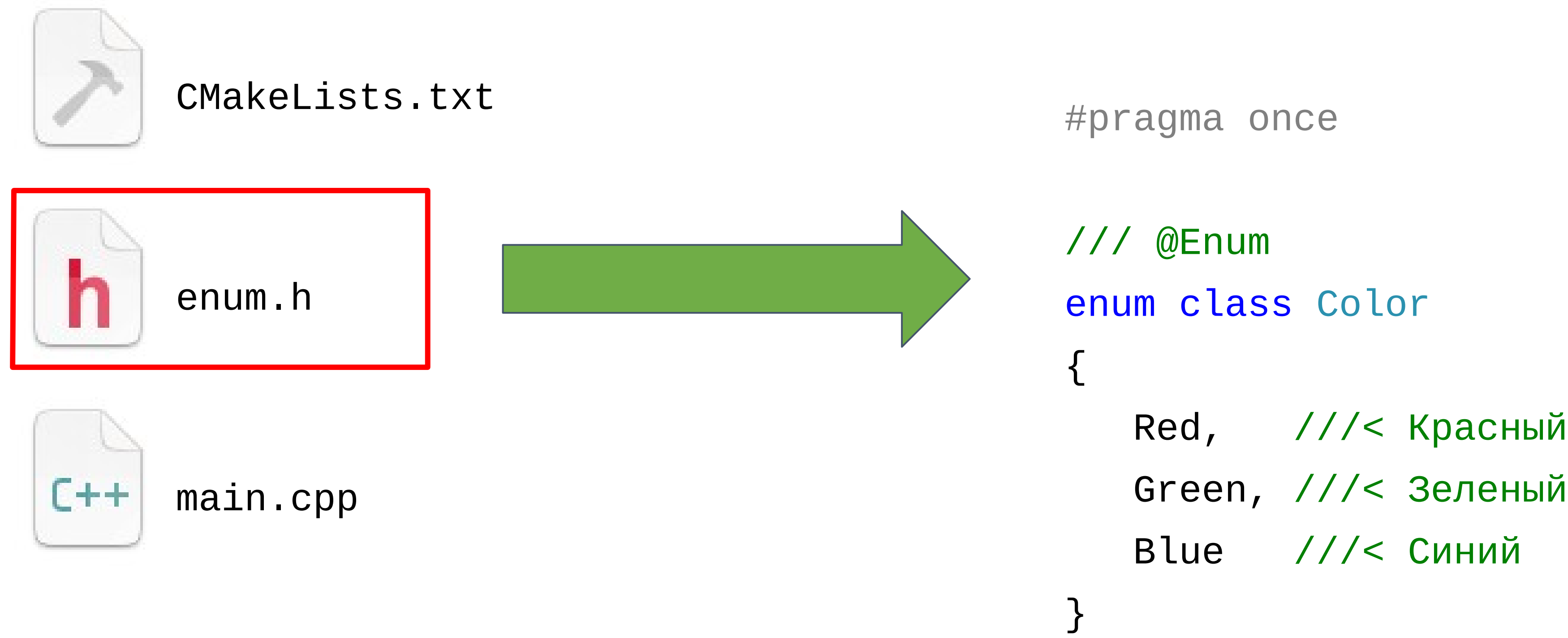
CMakeLists.txt

enum.h



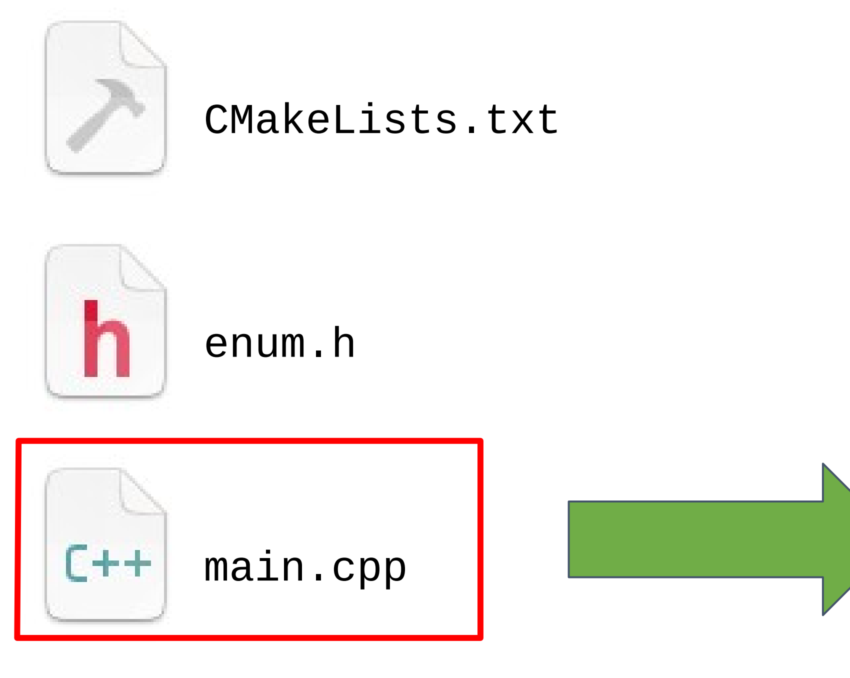
main.cpp

Внутреннее устройство




```cpp
#pragma once

/// @Enum
enum class Color
{
    Red,    ///< Красный
    Green,  ///< Зеленый
    Blue    ///< Синий
}
```

Внутреннее устройство

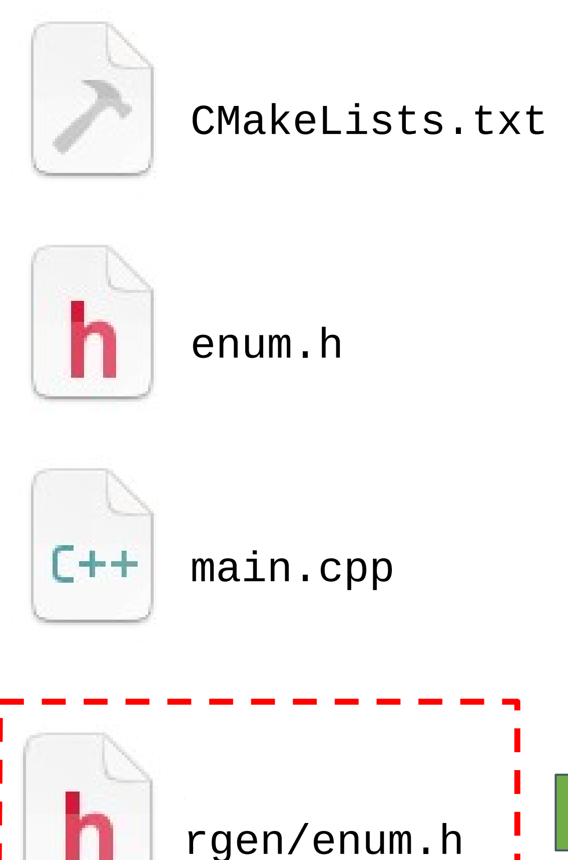



```cpp
#include "enum.h"
#include <rgen/enum.h>

int main()
{
    auto str = rgen::Enum<Color>::toString(Color::Red);
    if (str == "Красный")
        return 0;
    else
        return 1;
}
```

Внутреннее устройство

```cpp
#include "enum.h"
#include <rgen/enum.h>

int main()
{
    auto str = rgen::Enum<Color>::toString(Color::Red);
    if (str == "Красный")
        return 0;
    else
        return 1;
}
```

Внутреннее устройство




```
namespace rgen
{

template <typename EnumType>
class Enum
{
public:
    static std::string toString(
        EnumType value
    );
};

} // namespace rgen
```

Внутреннее устройство

```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

   // Однострочный cpp с #include "header"
   fakeCpp = getFakeCpp(header);

   // Сами формируем compile_commands.json
   cc = getCompileCommands(fakeCpp);

   // Анализ исходного кода, генерация доп. файлов
   rgen(cc, templ);

   // cmake подождет генерации зависимости таргетов
   // и на линковке ничего не потеряется
   compile_and_link()
}
```

Внутреннее устройство

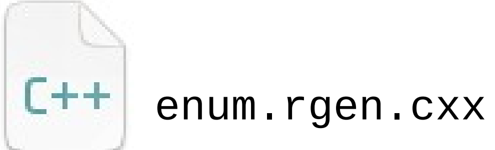
```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

   // Однострочный cpp с #include "header"
   fakeCpp = getFakeCpp(header);

   // Сами формируем compile_commands.json
   cc = getCompileCommands(fakeCpp);

   // Анализ исходного кода, генерация доп. файлов
   rgen(cc, templ);

   // cmake подождет генерации зависимости таргетов
   // и на линковке ничего не потеряется
   compile_and_link()
}
```

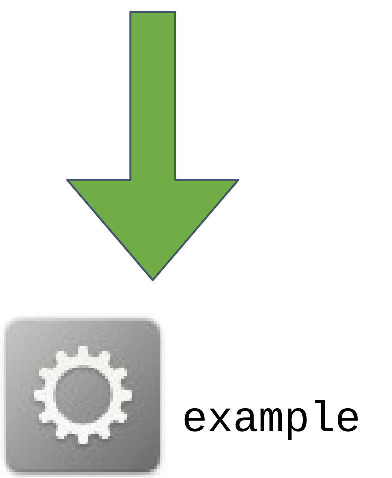
Псевдокод!

Иллюстрация последовательности действий

Внутреннее устройство

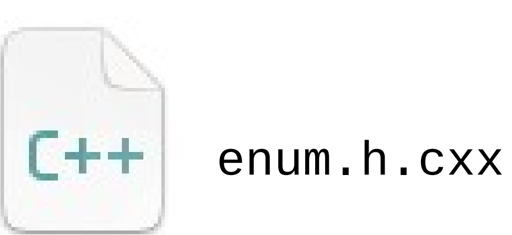
```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

}
```





Внутреннее устройство

```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

   // Однострочный cpp с #include "header"
   fakeCpp = getFakeCpp(header);









}
```



enum.h.cxx

```
#include "/home/uzver/example/enum.h"
```

Внутреннее устройство

```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

   // Однострочный cpp с #include "header"
   fakeCpp = getFakeCpp(header);

   // Сами формируем compile_commands.json
   cc = getCompileCommands(fakeCpp);

}
```

compile_commands.json

```
[{
 "arguments": [
     "/usr/bin/clang++",
     "-c",
     "-std=c++20",
     "-I/tools/gcc/include/c++/13",
     "--target=x86_64-pc-linux-gnu",
     "-x",
     "c++",
     "/example/build/example/enum.h.cxx"
 ],
 "file": "/example/build/example/enum.h.cxx"
}]
```

Внутреннее устройство

```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

   // Однострочный cpp с #include "header"
   fakeCpp = getFakeCpp(header);

   // Сами формируем compile_commands.json
   cc = getCompileCommands(fakeCpp);

   // Анализ исходного кода, генерация доп. файлов
   rgen(cc, templ);



}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                  templ = injaTemplate)
{
   // Вся информация компилятора об исходном коде
   astContext = ClangTool(cc).context;

   // Ходим по C++ AST дереву, собираем нужное
   extractedData = AstVisitor(astContext).data;

   // json под шаблон из инфы об исходном коде
   jsonPrintData = Converter(extractedData);

   // Формируем содержимое файла с нужной реализацией
   fileContent = inja::render(jsonPrintData, templ);
   writeFile(fileContent);
}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                templ = injaTemplate)
{

}
```

```
template<>
std::string Enum<Color>::toString(
  Color value
) {
  if(value == Color::Red)  {
    return "Красный";
  }
  if(value == Color::Green) {
    return "Зеленый";
  }
  if(value == Color::Blue) {
    return "Синий";
  }
}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                  templ = injaTemplate)
{

}
```

enum.cxx.j2

```
{% for enum in enums %}

template<>
std::string Enum<{{ enum.name }}>::toString(
   {{ enum.name }} value
) {
{%- for val in enum.values -%}
   if(value == {{ enum.name }}::{{ val.name }})
   {
       return "{{ value.display }}";
   }
{%- endfor -%}
}
{% endfor %}
```

Внутреннее устройство

```json
{
 "enums": [
   {
     "name": "Color",
     "values": [
       { "name": "Red",
         "display": "Красный" },
       { "name": "Green",
         "display": "Зеленый" },
       { "name": "Blue",
         "display": "Синий" }
     ]
   }
 ]
}
```

+

enum.cxx.j2

```
{% for enum in enums %}

template<>
std::string Enum<{{ enum.name }}>::toString(
   {{ enum.name }} value
) {
{%- for val in enum.values -%}
   if(value == {{ enum.name }}::{{ val.name }})
   {
       return "{{ value.display }}";
   }
{%- endfor -%}
}
{% endfor %}
```

Внутреннее устройство

```json
{
 "enums": [
   {
     "name": "Color",
     "values": [
       { "name": "Red",
         "display": "Красный" },
       { "name": "Green",
         "display": "Зеленый" },
       { "name": "Blue",
         "display": "Синий" }
     ]
   }
 ]
}
```

+

enum.cxx.j2

```
{% for enum in enums %}

template<>
std::string Enum<{{ enum.name }}>::toString(
   {{ enum.name }} value
) {
{%- for val in enum.values -%}
   if(value == {{ enum.name }}::{{ val.name }})
   {
       return "{{ value.display }}";
   }
{%- endfor -%}
}
{% endfor %}
```

Внутреннее устройство

```json
{
 "enums": [
   {
      "name": "Color",
      "values": [
        { "name": "Red",
          "display": "Красный" },
        { "name": "Green",
          "display": "Зеленый" },
        { "name": "Blue",
          "display": "Синий" }
      ]
   }
 ]
}
```

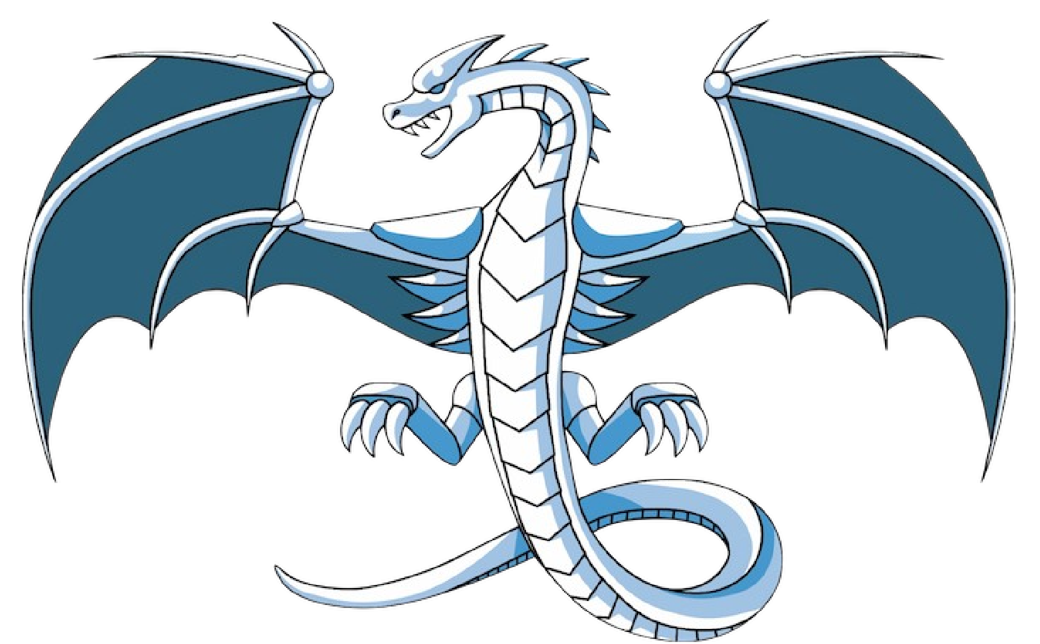
enum.cxx.j2

```
{% for enum in enums %}

template<>
std::string Enum<{{ enum.name }}>::toString(
   {{ enum.name }} value
) {
{%- for val in enum.values -%}
   if(value == {{ enum.name }}::{{ val.name }})
   {
       return "{{ value.display }}";
   }
{%- endfor -%}
}
{% endfor %}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                  templ = injaTemplate)
{
   // Вся информация компилятора об исходном коде
   astContext = ClangTool(cc).context;
}
```

Кормим дракона исходниками



owned by: ©Apple Inc, source, license: Non-exclusive, royalty-free license

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                  templ = injaTemplate)
{
   // Вся информация компилятора об исходном коде
   astContext = ClangTool(cc).context;

   // Ходим по C++ AST дереву, собираем нужное
   extractedData = AstVisitor(astContext).data;

}
```

```
#include <clang/AST/RecursiveASTVisitor.h>

struct Visitor
  : clang::RecursiveASTVisitor<Visitor>
{
// ...
}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                 templ = injaTemplate)
{
   // Вся информация компилятора об исходном коде
   astContext = ClangTool(cc).context;

   // Ходим по C++ AST дереву, собираем нужное
   extractedData = AstVisitor(astContext).data;



}
```

```
bool VisitEnumDecl(clang::EnumDecl* decl)
{
 if (comments(m_ast, *decl).Contains("Enum"))
   m_extractedData.enumDeclList.push_back(decl);

 return true;
}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                 templ = injaTemplate)
{
   // Вся информация компилятора об исходном коде
   astContext = ClangTool(cc).context;

   // Ходим по C++ AST дереву, собираем нужное
   extractedData = AstVisitor(astContext).data;

   // json под шаблон из инфы об исходном коде
   jsonPrintData = Converter(extractedData);



}
```

Внутреннее устройство

```cpp
inja::json converter() {
  inja::json res;
  for (const auto& enumDecl : extractedData.declList)
  {
    auto& enumJson = res["enums"].emplace_back();
    enumJson["name"] = enumDecl->getQualifiedNameAsString();
    for (auto* constantDecl : enumDecl->enumerators())
    {
      auto& valueJson = enumJson["values"].emplace_back();
      valueJson["name"] = constantDecl->getNameAsString();
      valueJson["display"] = displayContent(constantDecl)
    }
  }
  return res;
}
```

```json
{
 "enums": [
   {
     "name": "Color",
     "values": [
       { "name": "Red",
         "display": "Красный" },
       { "name": "Green",
         "display": "Зеленый" },
       { "name": "Blue",
         "display": "Синий" }
     ]
   }
 ]
}
```

Внутреннее устройство

```
void rgen(cc = compile_commands.json,
                  templ = injaTemplate)
{
   // Вся информация компилятора об исходном коде
   astContext = ClangTool(cc).context;

   // Ходим по C++ AST дереву, собираем нужное
   extractedData = AstVisitor(astContext).data;

   // json под шаблон из инфы об исходном коде
   jsonPrintData = Converter(extractedData);

   // Формируем содержимое файла с нужной реализацией
   fileContent = inja::render(jsonPrintData, templ);
   writeFile(fileContent);
}
```

```
template<>
std::string Enum<Color>::toString(
  Color value
) {
  if(value == Color::Red)  {
    return "Красный";
  }
  if(value == Color::Green) {
    return "Зеленый";
  }
  if(value == Color::Blue) {
    return "Синий";
  }
}
```

Внутреннее устройство

```
void main(header = annotatedHeader)
{  // Мы внутри сборки приложения

   // Однострочный cpp с #include "header"
   fakeCpp = getFakeCpp(header);

   // Сами формируем compile_commands.json
   cc = getCompileCommands(fakeCpp);

   // Анализ исходного кода, генерация доп. файлов
   rgen(cc, templ);

   // cmake подождет генерации за счет зависимости таргетов
   // и на линковке ничего не потеряется
   compile_and_link()
}
```

Своя ошибка этапа линковки

Своя ошибка этапа линковки

Источник: общественное достояние

Своя ошибка этапа линковки

```
void report_if_you_see_link_error_with_this_function() noexcept;

template <class T>
constexpr T unsafe_declval() noexcept {
   report_if_you_see_link_error_with_this_function();

   // ...
}
```

Boost.PFR

Своя ошибка этапа линковки

```
// ...Пояснения по устранению ошибки
void _________________RGEN_ERROR_FIND_ME_________________();

template <typename EnumType>
class Enum
{
public:
   static std::string toString(EnumType value)
   {
       _________________RGEN_ERROR_FIND_ME_________________();
   }
};
```

Своя ошибка этапа линковки

```
// ...Пояснения по устранению ошибки
void _________________RGEN_ERROR_FIND_ME_________________();

template <typename EnumType>
class Enum
{
public:
   static std::string toString(EnumType value)
   {
      _________________RGEN_ERROR_FIND_ME_________________();
   }
};
```

Изображение (пингвин tux) сгенерировано https://chatgpt.com/

Своя ошибка этапа линковки

```
// ...Пояснения по устранению ошибки
void _________________RGEN_ERROR_FIND_ME_________________();

template <typename EnumType>
class Enum
{
public:
   static std::string toString(EnumType value)
   {
       _________________RGEN_ERROR_FIND_ME_________________();
   }
};
```

Своя ошибка этапа линковки

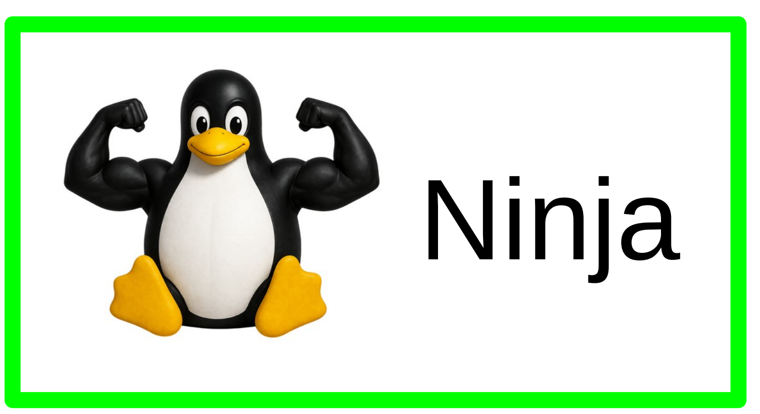
```
// ...Пояснения по устранению ошибки
struct _________________RGEN_ERROR_FIND_ME_________________ {};
using LinkErr =
    _________________RGEN_ERROR_FIND_ME_________________;

template <typename EnumType,
          typename LinkErrT = LinkErr>
class Enum
{
public:
   static std::string toString(EnumType value);
};
```

Выводим свой compile_commands.json

CMAKE_EXPORT_COMPILE_COMMANDS

author:the CMake team, source, license: CC BY 2.0

Выводим свой compile_commands.json

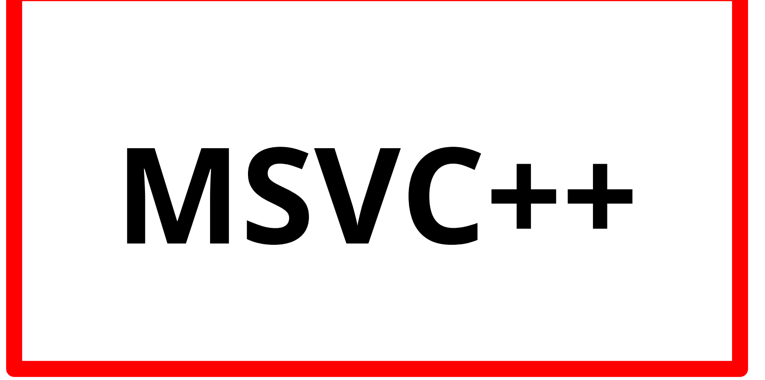


CMAKE_EXPORT_COMPILE_COMMANDS

Выводим свой compile_commands.json

Ninja

CMAKE_EXPORT_COMPILE_COMMANDS



Redkit Lab

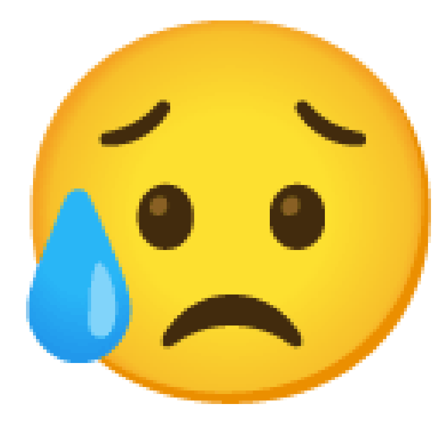
Выводим свой compile_commands.json

1. System includes
2. Include paths
3. Defines
4. CxxFlags
5. Library paths
6. ...

```json
[{
 "arguments": [
      "/usr/bin/clang++",
      "-c",
      "-std=c++20",
      "-I/tools/gcc/include/c++/13",
      "--target=x86_64-pc-linux-gnu",
      "-x",
      "c++",
      "/example/build/example/enum.h.cxx"
 ],
 "file": "/example/build/example/enum.h.cxx"
}]
```

Отношение как к макросам

```
#define CHECK_CONDITION(cond, msg)                                    \
   if (!(cond)) {                                                    \
       std::cerr << "Assert failed: " << #cond                       \
                 << " (" << msg << ") at " << __FILE__               \
                 << ":" << __LINE__ << " in " << __FUNCTION__        \
                 << std::endl;                                       \
       std::abort();                                                 \
   }
```



Отношение как к макросам

```cpp
void CheckConditionImpl(bool cond, const char* cond_str,
                        const char* msg, const char* file,
                        int line, const char* func) {
   if (!cond) {
       std::cerr << "Assert failed: " << cond_str
                 << " (" << msg << ") at " << file
                 << ":" << line << " in " << func << std::endl;
       std::abort();
   }
}

#define CHECK_CONDITION(cond, msg) \
   CheckConditionImpl(!!(cond), #cond, msg, __FILE__, __LINE__, __FUNCTION__)
```

Отношение как к макросам

```
/// @DoSomeShit
struct Point
{
    double x = {};
    double y = {};
}


template <typename T>
class DoSomeShit
{
public:
    static void doIt(T& obj);
};
```

Отношение как к макросам

```
/// @DoSomeShit
struct Point
{
    double x = {};
    double y = {};
}


template <typename T>
class DoSomeShit
{
public:
    static void doIt(T& obj);
};
```

```
{% for class in classes %}

template<>
void DoSomeShit<{{ class.name }}>::doIt(
    {{ class.name }}& obj) {
{%- for field in class.fields -%}
    _do();
    auto tmp = some(obj.{{field.name}});
    shit(tmp);
{%- endfor -%}
}

{% endfor %}
```

Отношение как к макросам

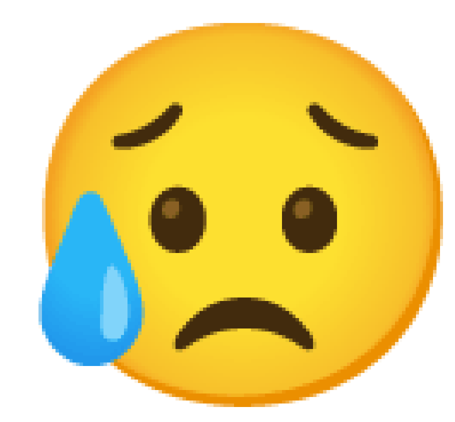
```
/// @DoSomeShit
struct Point
{
    double x = {};
    double y = {};
}


template <typename T>
class DoSomeShit
{
public:
    static void doIt(T& obj);
};
```

```
{% for class in classes %}

template<>
void DoSomeShit<{{ class.name }}>::doIt(
    {{ class.name }}& obj) {
{%- for field in class.fields -%}
    _do();
    auto tmp = some(obj.{{field.name}});
    shit(tmp);
{%- endfor -%}
}

{% endfor %}
```

Отношение как к макросам

```cpp
/// @DoSomeShit
struct Point
{
    double x = {};
    double y = {};
}


template <typename T>
class DoSomeShit
{
public:
    static void doIt(T& obj);
};
```

```
{% for class in classes %}

template<>
void DoSomeShit<{{ class.name }}>::doIt(
    {{ class.name }}& obj) {
{%- for field in class.fields -%}
{
    _do();
    auto tmp = some(obj.{{field.name}});
    shit(tmp);
}
{%- endfor -%}
}

{% endfor %}
```



Отношение как к макросам

```
template <typename T>
void doSomeShitInternal(
    T& field
) {
  _do();
  auto tmp = some(field);
  shit(tmp);
}
```

```
{% for class in classes %}

template<>
void DoSomeShit<{{ class.name }}>::doIt(
   {{ class.name }}& obj) {
{%- for field in class.fields -%}
   doSomeShitInternal(obj.{{field.name}});
{%- endfor -%}
}

{% endfor %}
```

Отношение как к макросам

```
/// @DoSomeShit
struct Point
{
    double x = {};
    double y = {};
}


template <typename T>
class DoSomeShit
{
public:
    static void doIt(T& obj);
};
```

```
template <>
void DoSomeShit<Point>::doIt(Point& obj) {
    doSomeShitInternal(obj.x);
    doSomeShitInternal(obj.y);
};
```

Нетривиальное тестирование

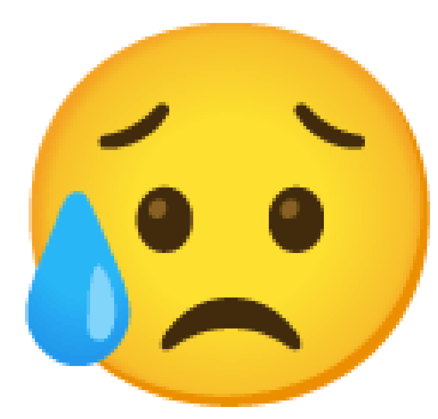
```
int someShit(int arg);

void test()
{
    EXPECT_EQ(someShit(5), 10);
    EXPECT_EQ(someShit(3), 6);
    EXPECT_EQ(someShit(6), 12);
    EXPECT_EQ(someShit(10), 99);
}
```

Нетривиальное тестирование

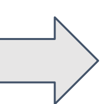
```
// Представим что определение функции генерируется
int someShit(int arg);

void test()
{
    EXPECT_EQ(someShit(5), 10);
    EXPECT_EQ(someShit(3), 6);
    EXPECT_EQ(someShit(6), 12);
    EXPECT_EQ(someShit(10), 99);
}
```

Нетривиальное тестирование

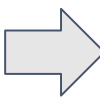
```
// Представим что определение функции генерируется
int someShit(int arg);

void test()
{
    EXPECT_EQ(someShit(5), 10);
    EXPECT_EQ(someShit(3), 6);
    EXPECT_EQ(someShit(6), 12);
    EXPECT_EQ(someShit(10), 99);
}
```



Меняем реализацию генерации



В генеренном коде ошибка компиляции



Тест не компилируется и его не отладить нормально

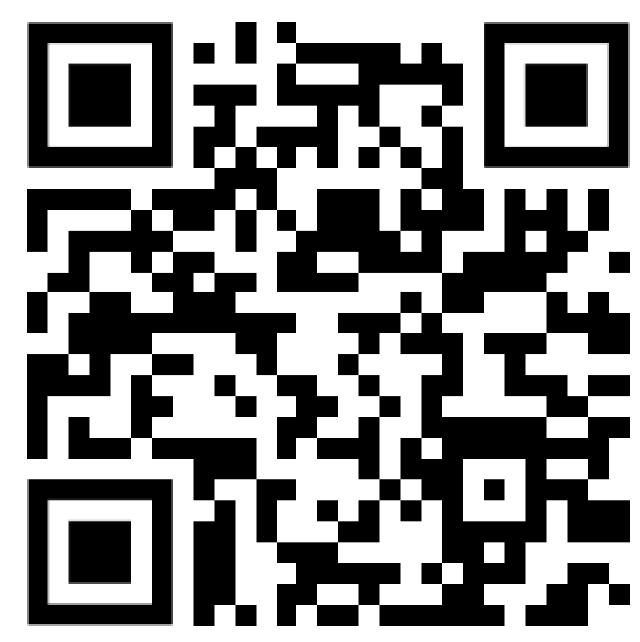
Выводы

1. Посмотрели много примеров самописной статической рефлексии. Концепции применимы и в стандартной рефлексии

2. Посмотрели, какие задачи из практики продуктовой разработки помогает проще решать рефлексия

3. Ждем, пока C++26 настоится

4. Неустанная борьба с boilerplate



rgen

Внимание!

Спасибо за внимание

Готов ответить на ваши вопросы

Спикер: Артем Бабаев
Должность: Руководитель группы
Компания: Redkit Lab
Почта: ivavpi@gmail.com
tg: @simplepersonru1
vk: id752278665

г. Москва, 2026 г.