Case Study: класс Config

На прошлой лекции

- Класс создает контекст для инварианта
- Конструктор устанавливает инвариант
- Инкапсуляция помогает поддерживать инвариант

Другие причины для создания класса:

- Упрощение передачи параметров в методы
- Ограничение влияния изменений
- Изоляция сложности
- и др.

Разберем задачу из ЛР2

```
ParsedText parse(string text, ???);
```

??? — параметры парсера:

- ngram length: min, max
- stop_words
- etc

Как передавать параметры в парсер?

config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<fts>
   <parser ngram_min_length="3" ngram_max_length="6">
      <!-- Или -->
      <ngram min="3" max="6"/>
      <stop_words>
         <word>a</word>
         <word>an</word>
         …
      </stop_words>
   </parser>
</fts>
```

config.json

```
{
  "fts": {
    "parser" {
      "ngrams": {
        "min": 3,
        "max": 6
      },
      "stop_words": ["a", "an", ... ]
    }
  }
}
```

Вариант 1: передавать разобранный документ

```
ParsedText parse(
  string text, pugi::xml_node parser_config);

ParsedText parse(string text, json parser_config);
```

Покритикуйте это решение

Оценка решения

? Решение кажется простым, не появляется новых сущностей

✗ Появляется зависимость функции parse от формата конфига

✗ Появляется транзитивная зависимость клиента функции parse от формата конфига

Вариант 2: отдельные переменные

```
ParsedText parse(
  string text,
  size_t ngram_min_len,
  size_t ngram_max_len,
  const StringSet& stop_words);
```

Покритикуйте это решение

Оценка решения

✓ Функция parse не зависит от формата конфига

✗ Функция неустойчива к изменению набора параметров парсера

¿ Где хранить параметры по умолчанию?
¿ Где проверять корректность диапазона (min, max)?

Вариант 3: выделенный тип

```
ParsedText parse(string text, ParserOpts parser_opts);
```

✓ Функция parse не зависит от формата конфига
✓ Функция parse устойчива к изменению набора параметров

¿ Появляется новая сущность (но проблема ли это?)

? Как реализовать **ParserOpts**?

Отступление: более общая картина

```
//! Конфиг всего поисковика
struct Config {
  ParserOpts parser_opts_;
  RankerOpts ranker_opts_;
  LimiterOpts limiter_opts_;
};

Config load_from_file(fs::path config_path);
Config load_from_string(std::string_view config_text);
Config load_from_stream(std::istream& is);
```

В интерфейсе

нет зависимости от формата

Вариант 3.1: структура

```
struct ParserOpts {
  std::size_t ngram_min_length_;
  std::size_t ngram_max_length_;
  StringSet stop_words_;
};
```

Нужен ли конструктор?

Вариант 3.1: структура

```
struct ParserOpts {
  std::size_t ngram_min_length_;
  std::size_t ngram_max_length_;
  StringSet stop_words_;
};
```

Есть ли инвариант?

Установка инварианта в конструкторе

```
ParserOpts::ParserOpts(std::size_t ngram_min_length,
                       std::size_t ngram_max_length,
                       StringSet stop_words)
    : ngram_min_length_(ngram_min_length),
      ... {
  if (ngram_min_length_ > ngram_max_length_) {
    throw std::domain_error("Invalid range");
  }
}
```

Локализация инварианта

Что обладает инвариантом?

- Весь блок настроек парсера?
- Один диапазон?

Локализация инварианта

```
ParserOpts::ParserOpts(Range ngram_range,
                       StringSet stop_words)
    : ngram_range_(ngram_range),
      stop_words_(std::move(stop_words) {
}
```

Локализация инварианта

```
struct Range {
  Range(std::size_t begin, std::size_t end)
      : begin_(begin), end_(end) {
    if (begin_ > end_) {
      throw std::domain_error("Invalid range");
    }
  }

  std::size_t begin_, end_;
}
```

Сколько можно плодить сущности?

Зависит от контекста

- ? Сколько еще полей в параметрах парсера?
- ? Есть ли другие места, где пригодится тип Range?
- 🎨 Как видит художник?

Вариант 3.2: структура с конструктором

```
struct ParserOpts {
  //! \throw std::domain_error on invalid range
  ParserOpts(...);

  std::size_t ngram_min_length_;
  std::size_t ngram_max_length_;
  StringSet stop_words_;
};
```

Пока остановимся на более плоском варианте

Вариант 3.3: защита инварианта

```
class ParserOpts {
 public:
  ParserOpts(...);

  std::size_t ngram_min_length() const { ... }
  ...

 private:
  std::size_t ngram_min_length_;
  ...
};
```

Все ок?

Вариант 3.3: защита инварианта

```
class ParserOpts {
 public:
  ParserOpts(...);

  std::size_t ngram_min_length() const { ... }
  ...

 private:
  std::size_t ngram_min_length_;
  ...
};
```

Кто видит проблему?

Допустим, параметров много (e.g. 30)

```
class ParserOpts {
 public:
  ParserOpts(...) : ... {}

  std::size_t ngram_min_length() const { ... }

 private:
  std::size_t ngram_min_length_;
  ...
};
```

30 параметров конструктора

30 полей в списке инициализации

30 геттеров

30 полей

Вариант 3.4: локальная защита инварианта

```
struct ParserOpts {
  class NgramRange {
   public:
    //! \throw std::domain_error on invalid range
    NgramRange(std::size_t min, std::size_t max);
    ...
  };

  NgramRange ngram_range_;
  StringSet stop_words_;
};
```

Выводы

- Есть широкий диапазон между конкретным и абстрактным.
- Задача разработчика — найти оптимум.

Примеры:

- ClangTidyOptions.h#L49
- ClangTidyOptions.cpp#L183
- clang/Format/Format.h#L55

Инициализация и копирование

Правило трех

Что есть в пустом классе?

```
struct Widget {









};
```

Что есть в пустом классе?

```
struct Widget {
  Widget() = default;

  ~Widget() = default;
  Widget(const Widget&) = default;
  Widget(Widget&&) = default;
  Widget& operator=(const Widget&) = default;
  Widget& operator=(Widget&& w) = default;
};
```

Что есть в пустом классе?

```
clang++ -std=c++20 -Xclang -ast-dump -fsyntax-only

CXXRecordDecl struct Widget definition
 |-DefaultConstructor
 |-CopyConstructor
 |-MoveConstructor
 |-CopyAssignment
 |-MoveAssignment
 `-Destructor
```


https://godbolt.org/z/bYW1TzqY8

Конструктор

Constructor is a **special** non-static member function of a class that is used to initialize objects of its class type.

Special. Very special.

constructor

Объявление функции

```
void f(int x);
```

1. Тип возвращаемого значения
2. Имя
3. Список параметров

Тип возвращаемого значения

```
void g() {}
void f() { return g(); } // Ok

class Widget {
 public:
  Widget() {
    return;       // Ok
    return g();   // Error
  }
};
```

Имя конструктора

```
class Widget {
 public:
  Widget() { ... }


  constructor() { ... }
};
```

Конструктор использует имя своего класса

Почему не выделили ключевое слово?

Указатель на функцию

```
void ff();

struct Widget {
  Widget() {}
  static void sf();
  void mf();
};
```

```
auto ffp = ff;
auto sfp = Widget::sf;
auto mfp = &Widget::mf;

ffp();
sfp();

Widget w;
(w.*mfp)();

auto cp = &Widget::Widget;
```


https://godbolt.org/z/YxT9ebbha

Порядок вызова специальных методов

Создание объекта:

1. Конструкторы полей в порядке определения
2. Конструктор класса

Уничтожение объекта:

1. Деструктор класса
2. Деструкторы полей в порядке, обратном определению

Этот порядок логичен, но возникают проблемы

https://godbolt.org/z/G4Wvjcecq

Инициализация констант и ссылок

```
struct Widget {
  Widget()
  {
    x_ = 42;
    y_ = x_;
    z_ = y_;
  }

  const int x_;
  int y_;
  int& z_;
};
```

Какие здесь ошибки?

Инициализация констант и ссылок

```
struct Widget {
  Widget()    // Error: uninitialized const int x_
  {           // Error: uninitialized int& z_
    x_ = 42;  // Error: assignment of read-only
    y_ = x_;
    z_ = y_;
  }

  const int x_;
  int y_;
  int& z_;
};
```

Инициализация констант и ссылок

```
struct Widget {
  Widget(): x_(42), y_(x_), z_(y_) {}

  const int x_;
  int y_;
  int& z_;
};
```

Инициализация конструктором с параметрами

```
class Widget {
 public:
  Widget() {
    t_ = Tracer(42);
  }

 private:
  Tracer t_;
};
```

Какие методы Tracer будут вызваны?

Инициализация конструктором с параметрами

```
class Widget {
 public:
  Widget() {
    t_ = Tracer(42);
  }

 private:
  Tracer t_;
};
```

```
Tracer()
Tracer(int)
operator=(Tracer&&)
~Tracer()
~Tracer()
```


https://godbolt.org/z/v957YY6T8

Инициализация конструктором с параметрами

```
class Widget {
 public:
  Widget() : t_(42) {
  }

 private:
  Tracer t_;
};
```

```
Tracer(int)
~Tracer()
```


https://godbolt.org/z/nEoTedW1W

Sleeping test

- Сколько может быть конструкторов в классе?
- Сколько может быть деструкторов?

Sleeping test

- Сколько может быть конструкторов в классе? — Сколько угодно.
- Сколько может быть деструкторов? — Только один.

Покритикуйте этот код

```
class MyString {
 public:
  MyString(const char* str)
      : length_(std::strlen(str)),
        str_(new char[length_ + 1]()) {
    std::copy(str, str + length_, str_);
  }
 private:
  ...
};
```

Покритикуйте этот код

```
class MyString {
 public:
  MyString(const char* str)
      : length_(std::strlen(str)),
        str_(new char[length_ + 1]()) {
    std::copy(str, str + length_, str_);
  }
 private:
  ...
};
```

Нет парного освобождения памяти

https://godbolt.org/z/3x6x4xorf

Деструктор

```
class MyString {
 public:

  ~MyString() {
    delete[] str_;
  }

 private:
  ...
};
```

Еще проблемы?

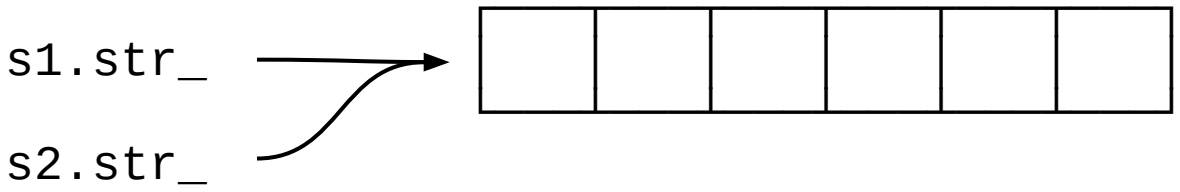
Что происходит в памяти?

```
MyString s1 = "Hello, World";

MyString s2(s1);
```

Что происходит в памяти?

```
MyString s1 = "Hello, World";
MyString s2(s1);
```



Конструктор копирования

```
class MyString {
 public:
  MyString(const MyString& other)
    : length_(other.length_),
      str_(new char[length_ + 1]()) {
    std::copy(other.str_, other.str_ + length_, str_);
  }
};
```

Устраняем дублирование

```
class MyString {
 public:
  MyString(const MyString& other)
     : MyString(other.str_) { }
};
```

Чуть хуже: два прохода по строке

Был бы полезен конструктор вида:

MyString(const char str, std::size_t length)*

https://godbolt.org/z/rK7hG8scx

N.B.

```
~MyString() {
  delete[] str_;
  str_ = nullptr;  // Нет смысла в занулении указателя
}
```

Что происходит в памяти?

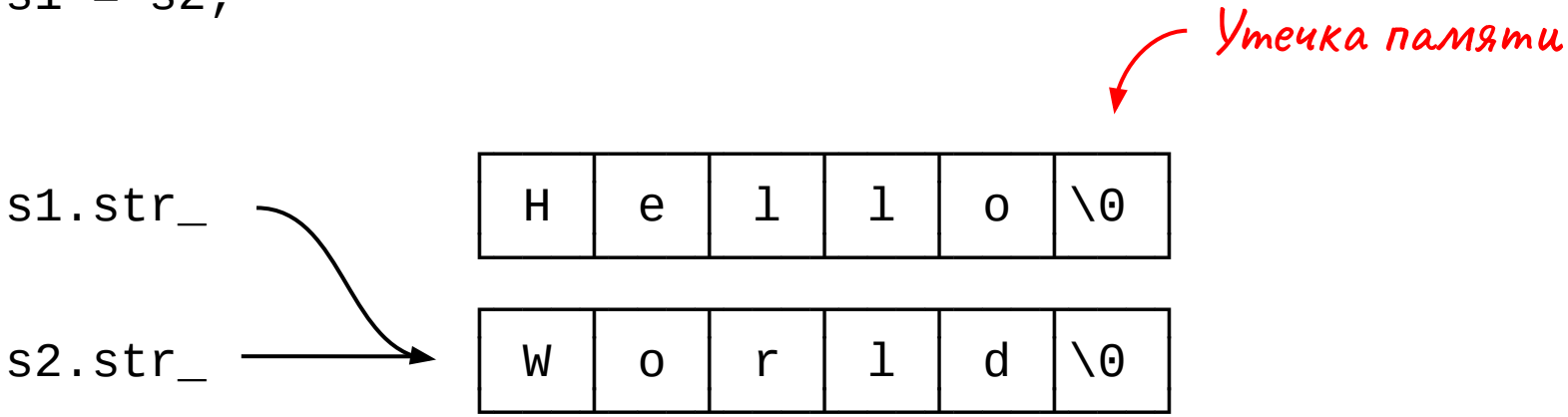
```
MyString s1 = "Hello”, s2 = “World";

s1 = s2;
```

Что происходит в памяти?

```
MyString s1 = "Hello”, s2 = “World";

s1 = s2;
```



https://godbolt.org/z/3YWdYeoa3

Copy Assignment

```
MyString& operator=(const MyString& other) {
  delete[] str_;
  str_ = new char[other.length_ + 1]();
  length_ = other.length_;
  std::copy(other.str_, other.str_ + length_, str_);
  str_[length_] = '\0';
  return *this;
}
```


https://godbolt.org/z/jnrYWhfWh

Copy Assignment

```
MyString& operator=(const MyString& other) {
  delete[] str_;
  str_ = new char[other.length_ + 1]();
  length_ = other.length_;
  std::copy(other.str_, other.str_ + length_, str_);
  str_[length_] = '\0';
  return *this;
}
```

Можем оптимизировать аллокации?

https://godbolt.org/z/jnrYWhfWh

Copy Assignment

```
MyString& operator=(const MyString& other) {
  if (length_ < other.length_) {
    delete[] str_;
    str_ = new char[other.length_ + 1]();
  }
  length_ = other.length_;
  std::copy(other.str_, other.str_ +length_, str_);
  str_[length_] = '\0';
  return *this;
}
```

TODO: capacity_

Copy Assignment

```
MyString& operator=(const MyString& other) {
  delete[] str_;
  str_ = new char[other.length_ + 1]();
  length_ = other.length_;
  std::copy(other.str_, other.str_ + length_, str_);
  str_[length_] = '\0';
  return *this;
}
```

Какие еще проблемы?

https://godbolt.org/z/jnrYWhfWh

Self Assignment

```
MyString s1 = "Hello";
s1 = s1;
std::cout << s1 << '\n';
```

Упражнение читателю: нарисуйте схему памяти

https://godbolt.org/z/nWYK7enb6

Self assignment

```
MyString& operator=(const MyString& other) {
  if (this == &other) {
    return *this;
  }
  ...
  return *this;
}
```


https://godbolt.org/z/19s3xoeME

Copy Assignment

- Реализованное присваивание все еще недостаточно хорошо
- Другой подход мы разберем в теме «Исключения»

C++03: Rule Of Three

Если в классе реализован хотя бы один из специальных методов:

- Деструктор
- Конструктор копирования
- Копирующий оператор присваивания

то следует реализовать все три.

Нужно еще два специальных метода

```
struct Widget {
  Widget() = default;

  ~Widget() = default;
  Widget(const Widget&) = default;
  Widget(Widget&&) = default;
  Widget& operator=(const Widget&) = default;
  Widget& operator=(Widget&& w) = default;
};
```

Converting Constructor

```
class Widget {
 public:
  /*explicit*/ Widget(int) {}
};

int main() {
  Widget w1(1);   // Ok, direct init
  Widget w2 = 1;  // Ok, copy init
}
```

Converting Constructor

```
class Widget {
 public:
  explicit Widget(int) {}
};

int main() {
  Widget w1(1);   // Ok, direct init
  Widget w2 = 1;  // Error, copy init
}
```


https://godbolt.org/z/ox7PT39P3

Converting Constructor

```
void f(Widget w) { ... }


// Ok, если конструктор explicit(false)
// Error, если конструктор explicit(true)
f(1);
```

Когда explicit(false) удивляет

```
class Vector {
 public:
  Vector(int size);
};

void f(Vector v);

f(42);
```

Когда explicit(false) удивляет

```
class Vector {
 public:
  explicit Vector(int size);
};

void f(Vector v);

f(42);  // Error
```

Когда explicit(false) удобен

```
class Rational {
 public:
  Rational(int num = 0, int denom = 1) { ... }
};

void f(Rational r);

f(42);
```

Рекомендация

C.46: By default, declare single-argument constructors explicit

Exception If you really want an implicit conversion from the constructor argument type to the class type, don’t use explicit

CppCoreGuidelines: C.46

Должен ли конструктор быть explicit?

```
class MyString {
 public:
  explicit(?) MyString(const char* str);

  ...
 private:
  ...
};
```

Как избежать аллокации?

```
void f(const MyString& s);


f("hello");
```

Как избежать аллокации?

```
void f(const MyString& s);
void f(const char* s);

f("hello");
```

Q&A