В предыдущей серии

Операторы сравнения

- Определяйте эти операторы только если тип наделен соответствующей семантикой
- C++17 и ранее:
  - Либо == и !=
  - Либо все шесть: ==, !=, <, <=, >, >=
- C++20:
  - Либо ==
  - Либо <=>
  - Либо <=> и == для оптимизации
- C++20 добавляет механизмы обращения и переписывания операторов

Функциональные объекты

Функция — не объект

- A function is not an object [...] [intro.object]
- Иногда возникает потребность использовать функцию как объект
- Указатель является объектом

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void*, const void*));
```

Указатель на функцию vs лямбда

```
bool int_less(int lhs, int rhs) { return lhs < rhs; }

// 1.3 times slower
std::sort(v.begin(), v.end(), int_less);

// 1.3 times faster
std::sort(v.begin(), v.end(),
          [](auto lhs, auto rhs) {
            return lhs < rhs;
          });
```


Quick Bench: function ptr vs lambda

Указатель на функцию vs лямбда

Пример синтетический: лямбда избыточна.

```
// 1.3 times faster
std::sort(v.begin(), v.end(),
          [](auto lhs, auto rhs) {
            return lhs < rhs;
          });
```

Сортировка чисел

```
std::vector v = {3, 1, 4, 1, 5, 9, 2, 6, 5};

// Использует оператор <
std::sort(v.begin(), v.end());
```

Сортировка составных объектов

```
struct Person {
  std::string first_name_;
  std::string last_name_;
  int age_;

  auto operator<=>(const Person& other) { ... }
};

// Использует пользовательский оператор <=>
std::sort(persons.begin(), persons.end());
```

Возможная реализация <=>

```
auto operator<=>(const Person& other) const {
  if (const auto last_name_order
                = last_name_ <=> other.last_name_;
      last_name_order != 0) {
    return last_name_order;
  }
  ...
}
```


https://godbolt.org/z/8435sdcxr

Возможная реализация <=>

```
auto operator<=>(const Person& other) const {
  using PersonView = std::tuple<std::string_view,
                                std::string_view,
                                int>;

  const auto lhs = PersonView(
      last_name_, first_name_, age_);
  const auto rhs = PersonView(other.last_name_, ...);
  return lhs <=> rhs;
}
```


https://godbolt.org/z/9EPzjbY6W

Проблема

- Пользовательский оператор определяет единственный критерий сортировки.
- В реальном приложении возможно множество компараторов.

C++03: Function objects

```
struct LastNameLess {
  bool operator()(const Person& lhs,
                  const Person& rhs) const {
    const int lnc
        = lhs.last_name_.compare(rhs.last_name_);
    if (lnc != 0)
      return lnc < 0;
    …
  }
};

std::sort(persons.begin(), persons.end(),
          LastNameLess());
```


https://godbolt.org/z/1G6he8KMa

Имитация замыканий (closure)

```
struct AddN {
  explicit AddN(int n)
    : n_(n) { }

  int operator()(int b) const {
    return n_ + b;
  }

  int n_;
};
```

Имитация замыканий (closure)

```
const AddN add_2(2);

const int x = add_2(40);

std::cout << "x = " << x << '\n';
```

Имитация замыканий (closure)

```
const AddN add_2(2);

std::vector numbers = {3, 1, 4, 1, 5, 9, 2, 6, 5};
std::vector<int> numbers_plus_2;

std::transform(
  numbers.begin(), numbers.end(),
  std::back_inserter(numbers_plus_2),
  add_2);
```

Мотивация лямбд

```
struct AddN {
  explicit AddN(int n)
    : n_(n) { }

  int operator()(int b) const {
    return n_ + b;
  }

  int n_;
};
```

Boilerplate

Полезный код

Эквивалентная лямбда

```
[n = 2](int b) -> int { return n + b; }
```

- `[]` — список захвата (capture list)
- `( )` — список параметров
- `-> T` — тип возвращаемого значения
- `{}` — тело функции

Эквивалентная лямбда

```
[n = 2](int b) -> int { return n + b; }
```

```
[n = 2](int b) { return n + b; }
```

- Тип возвращаемого значения может быть выведен
- `operator()` по умолчанию const
- Что внутри: https://cppinsights.io/s/f5a2416d

Захват по значению

```
int x = 42;

[m_x = x]() { return m_x + 1; };  // ≈ auto m_x = x

// 🔍
struct lambda {
  int m_x;
  lambda(int& x) : m_x(x) {}
  ...
};
```

Захват по значению

```
int x = 42;

[x = x]() { return x + 1; };

// 🔍
struct lambda {
  int x;
  lambda(int& _x) : x(_x) {}
  ...
};
```

Допустимо, но избыточно

Захват по значению

```
int x = 42;

[x]() { return x + 1; };

// 🔍
struct lambda {
  int x;
  lambda(int& _x) : x(_x) {}
  ...
};
```

Захват с тем же именем

Захват по ссылке

```
int x = 42;

[&m_x = x]() { return m_x + 1; };  // ≈ auto &m_x = x

// 🔍
struct lambda {
  int& m_x;
  lambda(int& x) : m_x(x) {}
  ...
};
```

Захват по ссылке под тем же именем

```
int x = 42;

[&x = x]() { return x + 1; };
[&x]() { return x + 1; };
// 🔍
struct lambda {
  int& x;
  lambda(int& _x) : x(_x) {}
  ...
};
```

Захват по значению: только используемое

```
int x = 42;

[=]() { return x + 1; };

// 🔍
struct lambda {
  int x;
  lambda(int& _x) : x(_x) {}
  ...
};
```

Захват по ссылке: только используемое

```
int x = 42;

[&]() { return x + 1; };

// 🔍
struct lambda {
  int& x;
  lambda(int& _x) : x(_x) {}
  ...
};
```

Захват по ссылке и по значению

```
int x = 40, y = 2;

[&x, y]() { return x + y; };

// 🔍
struct lambda {
  int& x;
  int y;
  lambda(int& _x, int& _y) : x(_x), y(_y)  {}
  ...
};
```

Захват глобальных переменных

`[&]` и `[=]` не захватывают глобальные переменные

Захват глобальных переменных

```
int g = 10;

int main() {
  auto kitten = [=]() { return g + 1; };
  auto cat = [g = g]() { return g + 1; };

  g = 20;

  std::cout << kitten() << '\n';
  std::cout << cat() << '\n';
}
```

Захват глобальных переменных

```
int g = 10;

int main() {
  auto kitten = [=]() { return g + 1; };
  auto cat = [g = g]() { return g + 1; };

  g = 20;

  std::cout << kitten() << '\n';  // 21
  std::cout << cat() << '\n';     // 11
}
```


https://godbolt.org/z/ooj8b4a44

Захват this

Ожидание:

```
[n = 42]() { std::cout << this->n << '\n'; };
```

Захват this

Ожидание:

```
[n = 42]() { std::cout << this->n << '\n'; };
```

Реальность:

```
error: 'this' was not captured for this lambda
function
```

Захват this: мотивация

```
struct Widget {
  void work(int x) { ... }

  void sync_work(int x) {
    this->work(x);
  }

  auto async_work(int x) {
    return std::async([=, this]() { this->work(x); });
  }
};
```


https://godbolt.org/z/PGKhbvjGo

Захват this: варианты

```
[=]() { this->work(); }       // Deprecated in C++20
[this]() { this->work(); }    // Ok
[&]() { this->work(); }       // Ok
[*this]() { this->work(); }   // Ok, C++17
```

Лямбда с состоянием

```
auto fib_gen = [a = 0, b = 1]() {
  const auto ret = a;
  a += b;
  std::swap(a, b);
  return ret;
};
```

В чем здесь проблема?

Лямбда с состоянием

```
auto fib_gen = [a = 0, b = 1]() {
  const auto ret = a;
  a += b;
  std::swap(a, b);
  return ret;
};
```

gcc: assignment of read-only variable 'a'

clang: cannot assign to a variable captured by copy in a non-mutable lambda

Лямбда с состоянием

```
auto fib_gen = [a = 0, b = 1]() {
  const auto ret = a;
  a += b;
  std::swap(a, b);
  return ret;
};

struct lambda { int operator() const; };
```

Лямбда с состоянием

```
auto fib_gen = [a = 0, b = 1]() mutable {
  const auto ret = a;
  a += b;
  std::swap(a, b);
  return ret;
};

struct lambda { int operator(); };
```


https://godbolt.org/z/4dfe84oah

Лямбда без захвата

```
auto less = [](int a, int b) { return a < b; };
```

Такая лямбда не содержит полей

- Внутренности: https://cppinsights.io/s/0538b86f
- Использование: https://godbolt.org/z/ejoG15qe5

Передача лямбды в функцию

```
struct Shelf {
  template <typename Function>
  void for_each(Function f) const {
    for (const auto& book : books_) {
      f(book);
    }
  }
  std::vector<Book> books_;
};
```


https://godbolt.org/z/7KjMMY1T1

Возврат лямбды

```
auto make_f() {
  return []() {
    return 42;
  };
}
```

Несемантичное использование ()

```
template <typename T>
class Matrix {
 public:
  const T& operator()(size_t i, size_t j) const;
  T& operator()(size_t i, size_t j);
};
```

Интерактив: лямбды и алгоритмы

https://godbolt.org/z/d1MrzMxY5

Материалы

Lambda expressions in C++

Back to Basics: Lambdas from Scratch - Arthur O'Dwyer - CppCon 2019

Вопросы?