Операторы сравнения C++17

TL;DR Операторы сравнения в C++17

- Базовые операторы: <, ==
- Прочие выражаются через базовые: <=, >, >=, !=
- Реализации <=, >, >=, != есть в пространстве имен std::rel_ops (Deprecated in C++20)

Пусть дан класс

```
struct Int {
  Int(int v): v_(v) {}
  int v_;
};
```

- Пока нам неважно сокрытие данных
- Пример еще проще, чем Rational

Операторы сравнения в C++17

```
bool operator< (Int lhs, Int rhs) { return lhs.v_ < rhs.v_; }
bool operator==(Int lhs, Int rhs) { return lhs.v_ == rhs.v_; }
```

Реализуем остальные операторы: https://godbolt.org/z/G7ade9cs6

Операторы сравнения в C++17

```
bool operator< (Int lhs, Int rhs) { return lhs.v_ < rhs.v_; }
bool operator<=(Int lhs, Int rhs) { return !(rhs < lhs); }
bool operator> (Int lhs, Int rhs) { return rhs < lhs; }
bool operator>=(Int lhs, Int rhs) { return !(lhs < rhs); }
bool operator==(Int lhs, Int rhs) { return lhs.v_ == rhs.v_; }
bool operator!=(Int lhs, Int rhs) { return !(lhs == rhs); }
```

Операторы сравнения в C++17

```
bool operator< (Int lhs, Int rhs) { return lhs.v_ < rhs.v_; }
bool operator<=(Int lhs, Int rhs) { return !(rhs < lhs); }
bool operator> (Int lhs, Int rhs) { return rhs < lhs; }
bool operator>=(Int lhs, Int rhs) { return !(lhs < rhs); }
bool operator==(Int lhs, Int rhs) { return lhs.v_ == rhs.v_; }
bool operator!=(Int lhs, Int rhs) { return !(lhs == rhs); }
```

Почему внешние функции, а не методы класса?

Проблемы операторов сравнения в C++17

1. Очевидно: Основные операторы < и ==,
   остальные выражаются через них.
   Мы хотели бы сократить количество кода.
2. Менее очевидно: на 6 строк кода 12 ошибок.
3. Совсем неочевидно: в некоторых случаях 6 операторов мало.

std::rel_ops (deprecated in C++20)

```
// You write
using namespace std::rel_ops;

// You get
template <typename T>
bool operator!=(const T& lhs, const T& rhs);
              >
              <=
              >=
```


std::rel_ops

Проблемы операторов сравнения в C++17

1. Очевидно: Основные операторы < и ==,
   остальные выражаются через них.
   Мы хотели бы сократить количество кода.
2. **Менее очевидно: на 6 строк кода 12 ошибок.**
3. Совсем неочевидно: в некоторых случаях 6 операторов мало.

Операторы должны быть constexpr и noexcept

```
constexpr bool operator< (Int lhs, Int rhs) noexcept { ... }
constexpr bool operator<=(Int lhs, Int rhs) noexcept { ... }
constexpr bool operator> (Int lhs, Int rhs) noexcept { ... }
constexpr bool operator>=(Int lhs, Int rhs) noexcept { ... }
constexpr bool operator==(Int lhs, Int rhs) noexcept { ... }
constexpr bool operator!=(Int lhs, Int rhs) noexcept { ... }
```

Конструктор Int тоже должен быть constexpr

Проблемы операторов сравнения в C++17

1. Очевидно: Основные операторы < и ==,
   остальные выражаются через них.
   Мы хотели бы сократить количество кода.
2. Менее очевидно: на 6 строк кода 12 ошибок.
3. **Совсем неочевидно: в некоторых случаях 6 операторов мало.**

Задача

- Вы разрабатываете класс String
- Сколько операторов сравнения нужно перегрузить и почему?

Считаем: 6 операторов по шаблону

```
bool operator< (const String& lhs, const String& rhs);
bool operator<=(const String& lhs, const String& rhs);
bool operator> (const String& lhs, const String& rhs);
bool operator>=(const String& lhs, const String& rhs);
bool operator==(const String& lhs, const String& rhs);
bool operator!=(const String& lhs, const String& rhs);
```

Конструктор String может выделять память

Мы не хотим аллокаций из-за неявных преобразований:

```
class String {
 public:
  String(const char* str):
    len_(strlen(str)),
    buf_(new char[len_]) { ... }
};
```

В реальности все несколько сложнее

Вызов оператора <

```
String s = "hello";
bool r1 = "world" < s;
bool r2 = s < "world";
```

Вызов оператора <

```
String s = "hello";
bool r1 = "world" < s;  // String("world") < s
bool r2 = s < "world";  // s < String("world")
```

Из-за неявного преобразования типов возникают аллокации

…еще 12 для сравнения с const char*

```
bool operator< (const char* lhs, const String& rhs);
bool operator< (const String& lhs, const char* rhs);
...
```

Это слишком много!

```
bool operator< (const String& lhs, const String& rhs);
bool operator<=(const String& lhs, const String& rhs);
bool operator> (const String& lhs, const String& rhs);
bool operator>=(const String& lhs, const String& rhs);
bool operator==(const String& lhs, const String& rhs);
bool operator!=(const String& lhs, const String& rhs);

bool operator< (const char* lhs, const String& rhs);
bool operator<=(const char* lhs, const String& rhs);
bool operator> (const char* lhs, const String& rhs);
bool operator>=(const char* lhs, const String& rhs);
bool operator==(const char* lhs, const String& rhs);
bool operator!=(const char* lhs, const String& rhs);

bool operator< (const String& lhs, const char* rhs);
bool operator<=(const String& lhs, const char* rhs);
bool operator> (const String& lhs, const char* rhs);
bool operator>=(const String& lhs, const char* rhs);
bool operator==(const String& lhs, const char* rhs);
bool operator!=(const String& lhs, const char* rhs);
```

Решим эту проблему на следующей лекции

Рекомендация

Перегружайте операторы сравнения для типов только если у них есть соответствующая семантика.

Контрпример

std::set и std::sort по умолчанию ожидают, что у типа есть оператор <

Необходимость хранить экземпляры класса в std::set или сортировать контейнер объектов — недостаточный повод для определения оператора < для вашего типа.

Решим эту задачу через одну лекцию

Q&A