
Яндекс

Яндекс

# Контракты C++

Antony Polukhin
Полухин Антон

Автор Boost библиотек TypeIndex, DLL, Stacktrace
Maintainer Boost Any, Conversion, LexicalCast, Variant
Представитель РГ21, ISO WG21 national body

# О чём поговорим?

# О чём поговорим

- Контракты – что это такое ?

# О чём поговорим

- Контракты – что это такое ?
- Алгоритмы из <algorithm>

# О чём поговорим

- Контракты – что это такое ?
- Алгоритмы из <algorithm>
- “O” большое

# О чём поговорим

- Контракты – что это такое ?
- Алгоритмы из <algorithm>
- “O” большое
- Контракты и алгоритмы

# Contracts / Контракты

# Contracts

Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции.

# Contracts — ассерты на стероидах

```
void push(int x, queue & q);
```

# Contracts — ассерты на стероидах

```
void push(int x, queue & q)
  [[expects: !q.full()]]
  [[ensures: !q.empty()]]
{
  //...
  [[assert: q.is_valid()]];
  //...
}
```

# Contracts — ассерты на стероидах

```
void push(int x, queue & q) [[expects: !q.full()]] [[ensures: !q.empty()]] ;

queue q;

// ...
```

# Contracts — ассерты на стероидах

```
void push(int x, queue & q) [[expects: !q.full()]] [[ensures: !q.empty()]] ;

queue q;

// ...

if (!q.full()) {
    push(10, q); // Хм... можно не проверять
    if (q.empty()) { // Хм...
    }
} else {
    push(11, q); // Хм... подозрительно
}
```

# Contracts — ассерты на стероидах

```
void(const std::contract_violation &);


namespace std {
  class contract_violation {
  public:
    int line_number() const noexcept;
    const char * file_name() const noexcept;
    const char * function_name() const noexcept;
    const char * comment() const noexcept;
  };
}
```

# <algorithm>

# std::sort(beg, end)

Отсортировать по возрастанию диапазон значений

| 4 | 6 | 9 | 1 | 2 | 5 | 3 | 8 | 7 | 0 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

# std::sort(beg, end)

Отсортировать по возрастанию диапазон значений

| 4 | 6 | 9 | 1 | 2 | 5 | 3 | 8 | 7 | 0 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

# std::sort(beg, end)

```
template <class RandomAccessIterator>
void sort(RandomAccessIterator beg, RandomAccessIterator end)
    [[expects: beg <= end]]
    [[ensures: is_sorted(beg, end)]]
;
```

# «О» большое

# «O» большое

“O” большое – время работы алгоритма/функции в зависимости от количества входных элементов N

# «O» большое

“O” большое – время работы алгоритма/функции в зависимости от количества входных элементов N

```
for (size_t i = 0; i < N; ++i)
```

=> O(N)

# «O» большое

“O” большое – время работы алгоритма/функции в зависимости от количества входных элементов N

```
for (size_t i = 0; i < N; ++i)                  => O(N)
for (size_t i = 0; i < N; ++i)
        for (size_t j = 0; j < N; ++j)          => O(N^2)
```

# «O» большое

| N | N*log(N) | N*N |
|---|---|---|
| 2 | 2 | 4 |
| 4 | 8 | 16 |
| 8 | 24 | 64 |
| 16 | 64 | 256 |
| 32 | 160 | 1,024 |
| 64 | 384 | 4,096 |
| 128 | 896 | 16,384 |
| 256 | 2,048 | 65,536 |
| 512 | 4,608 | 262,144 |
| 1,024 | 10,240 | 1,048,576 |

# «O» большое

- std::sort => $O( N*\log_2(N) )$

# «O» большое

- std::sort => $O( N*\log_2(N) )$
- std::stable_sort => $O( N*\log_2^2(N) )$

# «O» большое

- ~~std::sort~~ => O( N*$\log_2$(N) )
- ~~std::stable_sort~~ => O( N*$\log_2^2$(N) )

- std::minmax_element => O( N )
- std::nth_element => ~ O( N )
- std::partial_sort => O( N*$\log_2$(S) )

# std::nth_element(beg, mid, end)

- Выставить значение по итератору mid так чтобы:
  - Если отсортировать [beg, end) то значение mid не изменится
  - Слева от mid – значения *большие* или *равные* mid
  - Справа от mid - значения *меньшие* или *равные* mid

| 4 | 0 | 3 | 1 | 2 | 5 | 9 | 8 | 7 | 6 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

# std::nth_element(beg, mid, end)

- Выставить значение по итератору mid так чтобы:
  - Если отсортировать [beg, end) то значение mid не изменится
  - Слева от mid – значения *большие* или *равные* mid
  - Справа от mid - значения *меньшие* или *равные* mid

| 4 | 0 | 3 | 1 | 2 | 5 | 9 | 8 | 7 | 6 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

# std::nth_element

Найти 5 людей с наименьшим балансом

```
std::nth_element(v.begin(), v.begin() + 4, v.end());
```

Найти 5 людей с наибольшим балансом

```
std::nth_element(v.begin(), v.begin() + 4, v.end(), std::greater<>{});
```

Найти 1001 позвонившего

```
std::nth_element(v.begin(), v.begin() + 1000, v.end());
```

# std::partial_sort(beg, mid, end)

- Выставить значение по итератору mid так чтобы:
  - [beg, mid) не изменятся, если отсортировать [beg, end)
  - [beg, mid) - отсортированы

| 0 | 1 | 2 | 3 | 4 | 9 | 5 | 8 | 7 | 6 | @ |
|---|---|---|---|---|---|---|---|---|---|---|

# std::partial_sort

Распределить 5 призовых мест по наименьшему кол-ву штрафных балов

```
std::partial_sort(v.begin(), v.begin() + 5, v.end());
```

Покарать 5 школьников, пришедших последними на урок

```
std::partial_sort(v.begin(), v.begin() + 5, v.end(), std::greater<>{});
```

# std::minmax_element

Найти самого бедного и самого богатого клиента банка

```
auto mm = std::minmax_element(v.begin(), v.end());
std::cout << *mm.first << ' ' << *mm.second << '\n';
```

Внимание. Вопрос:

Как получить сортированный список из 10 человек с балансом на счету близким с к медиане?

(Как отсортировать всех по балансу и выбрать 10 человек из серединки).

# Как?

```
auto it = v.begin() + v.size() / 2 - 5;
```

# Как?

```
auto it = v.begin() + v.size() / 2 - 5;
const auto f = [](const auto& v1, const auto& v2) {
    return v1.balance() < v2.balance();
};
```

# Как?

```
auto it = v.begin() + v.size() / 2 - 5;
const auto f = [](const auto& v1, const auto& v2) {
    return v1.balance() < v2.balance();
};

std::nth_element(v.begin(), it, v.end(), f);
```

# Как?

```
auto it = v.begin() + v.size() / 2 - 5;
const auto f = [](const auto& v1, const auto& v2) {
    return v1.balance() < v2.balance();
};

std::nth_element(v.begin(), it, v.end(), f);
std::partial_sort(it + 1, it + 10, v.end(), f);
```

# O(N*log(10) ) vs O(N*log(N) )

| N | N*log(10)<br>$N+(N/2-1)*\log_2(9)$ | N*log(N) | N*log(N) – N*log(10) |
|---|---|---|---|
| 10 | 33 | 33 | 0 |
| 16 | 53 | 64 | 11 |
| 512 | 1,701 | 4,608 | 2,907 |
| 16,384 | 54,426 | 229,376 | 174,950 |
| 524,288 | 1,741,647 | 9,961,472 | 8,219,825 |
| 16,777,216 | 55,732,705 | 402,653,184 | 346,920,479 |

Как бы так сделать, чтобы предыдущую задачу решал компилятор а не мы?

# ???

```
template <class RandomAccessIterator>
void nth_element(RandomAccessIterator beg, RandomAccessIterator mid,
RandomAccessIterator end)
    [[expects: beg <= mid]]    [[expects: mid <= end]]
    [[ensures: ??? ]]
;
template <class RandomAccessIterator>
void partial_sort(RandomAccessIterator beg, RandomAccessIterator mid,
RandomAccessIterator end)
    [[expects: beg <= mid]]    [[expects: mid <= end]]
    [[ensures: ??? ]]
;
```

# Контракты и алгоритмы

# Итого

Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции

# Итого

- Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции:
  - Единый синтаксис описания области определения

# Итого

- Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции:
  - Единый синтаксис описания области определения
  - Долгий путь к формальному доказательству корректности программы

# Итого

- Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции:
  - Единый синтаксис описания области определения
  - Долгий путь к формальному доказательству корректности программы
  - Возможности подсказывать компилятору оптимизации

# Итого

- Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции:
  - Единый синтаксис описания области определения
  - Долгий путь к формальному доказательству корректности программы
  - Возможности подсказывать компилятору оптимизации
  - Много проблем

# Итого

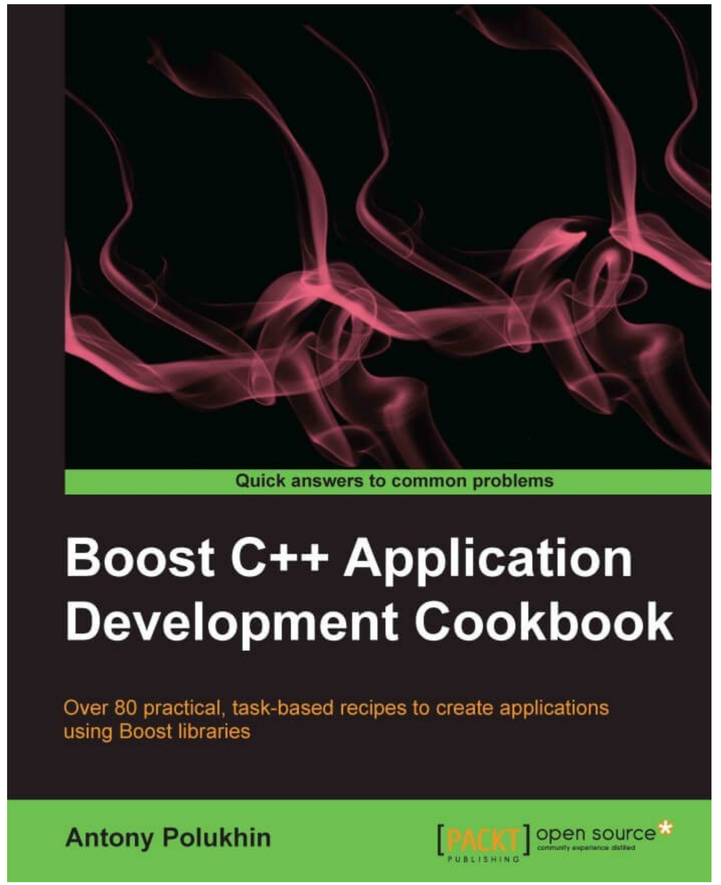
- Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции:
  - Единый синтаксис описания области определения
  - Долгий путь к формальному доказательству корректности программы
  - Возможности подсказывать компилятору оптимизации
  - Много проблем
  - Много мест для исследований :-)

# Спасибо! Вопросы?

https://stdcpp.ru

C++

РГ21 C++ РОССИЯ




http://apolukhin.github.io/Boost-Cookbook

# Итого

- Контракты – будущая возможность языка C++, позволяющая описать компилятору область определения функции:
  - Единый синтаксис описания области определения