

Яндекс

Как за час сделать недельную работу

Antony Polukhin
Полухин Антон

Автор Boost библиотек TypeIndex, DLL, Stacktrace
Maintainer Boost Any, Conversion, LexicalCast, Variant
Представитель PГ21, ISO WG21 national body

О чём поговорим?

О чём поговорим

Полу-готовые решения в C++

О чём поговорим

- Полу-готовые решения в C++
- Ускоряем программу в пару шагов

Стандартная библиотека



Работает как нам надо из-за:

Работает как нам надо из-за:

Шаблоны

Работает как нам надо из-за:

Шаблоны

```
std::vector<int> ign0;
```

Работает как нам надо из-за:

Шаблоны

```
std::vector<int> ign0;
```

```
struct unpredictable_thing;
```

```
std::vector<unpredictable_thing> ign1;
```

Работает как нам надо из-за:

Шаблоны

ADL

Работает как нам надо из-за:

Шаблоны

ADL:

```
namespace qwe_qwe_bar {
```

```
struct my_mega_type_thing;
```

```
} // namespace qwe_qwe_bar
```

Работает как нам надо из-за:

Шаблоны

ADL:

```
namespace qwe_qwe_bar {  
  
struct my_mega_type_thing;  
  
inline std::ostream& operator<<(  
    std::ostream& os,  
    const my_mega_type_thing& v);  
  
} // namespace qwe_qwe_bar
```

Работает как нам надо из-за:

Шаблоны

ADL:

```
namespace ololo {  
  
void adl_example() {  
    std::cout << qwe_qwe_bar::my_mega_type_thing();  
}  
  
} // namespace ololo
```

Работает как нам надо из-за:

Шаблоны

ADL

Traits и прочее

Работает как нам надо из-за:

Шаблоны

ADL

Traits и прочее <=====

char_traits



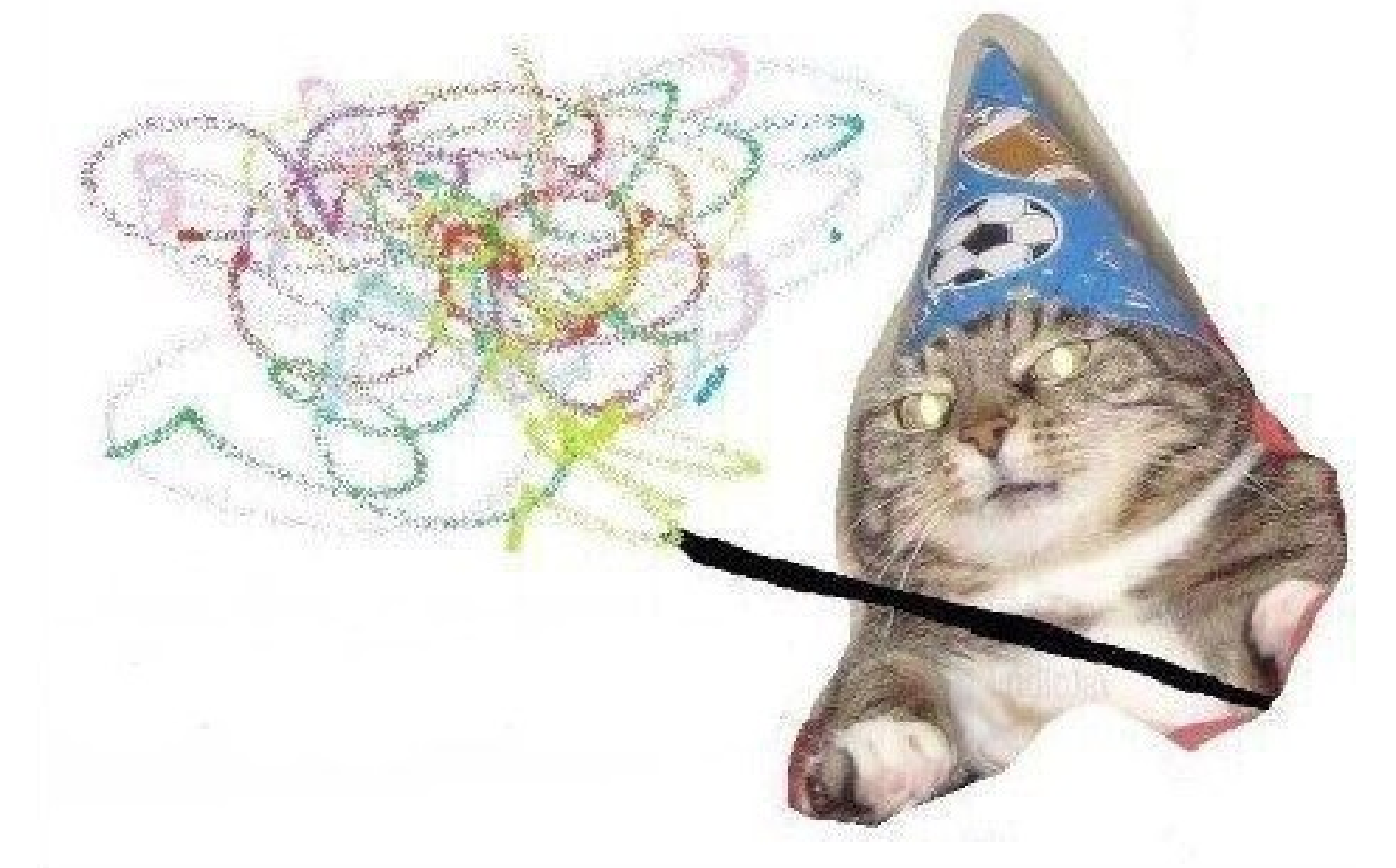
Хранить строки в нижнем регистре

Хранить в нижнем регистре

```
template <class Char>
struct lchar_traits : public std::char_traits<Char> {
    static void assign(Char& c1, const Char& c2) { c1 = std::tolower(c2); }

    static Char* move(Char* s1, const Char* s2, std::size_t n) {
        for (std::size_t i = 0; i < n; ++i)
            assign(s1[i], s2[i]);
        return s1;
    }
}
```

```
static Char* copy(Char* s1, const Char* s2, size_t n) { return move(s1, s2, n); }
```



Хранить в нижнем регистре

```
template <class Char>  
using lbasic_string = std::basic_string<Char, lchar_traits<Char> >;  
  
using lstring = lbasic_string<char>;  
using lwstring = lbasic_string<wchar_t>;
```

Хранить в нижнем регистре

```
template <class Char>
using lbasic_string = std::basic_string<Char, lchar_traits<Char> >;

using lstring = lbasic_string<char>;
using lwstring = lbasic_string<wchar_t>;

template <class Char, class Traits, class Char2>
std::basic_ostream<Char, Traits>& operator<<(
    std::basic_ostream<Char, Traits>& os, const lbasic_string<Char2>& str)
{ return os.write(str.data(), str.size()); }
```

Хранить в нижнем регистре

```
lstring s1 = "Hello";  
lstring s2 = "heLLo";  
if (s1 == s2)  
    std::cout << s1 << " and " << s2 << " are equal\n";
```

Выведет: hello and hello are equal

Хранить в нижнем регистре

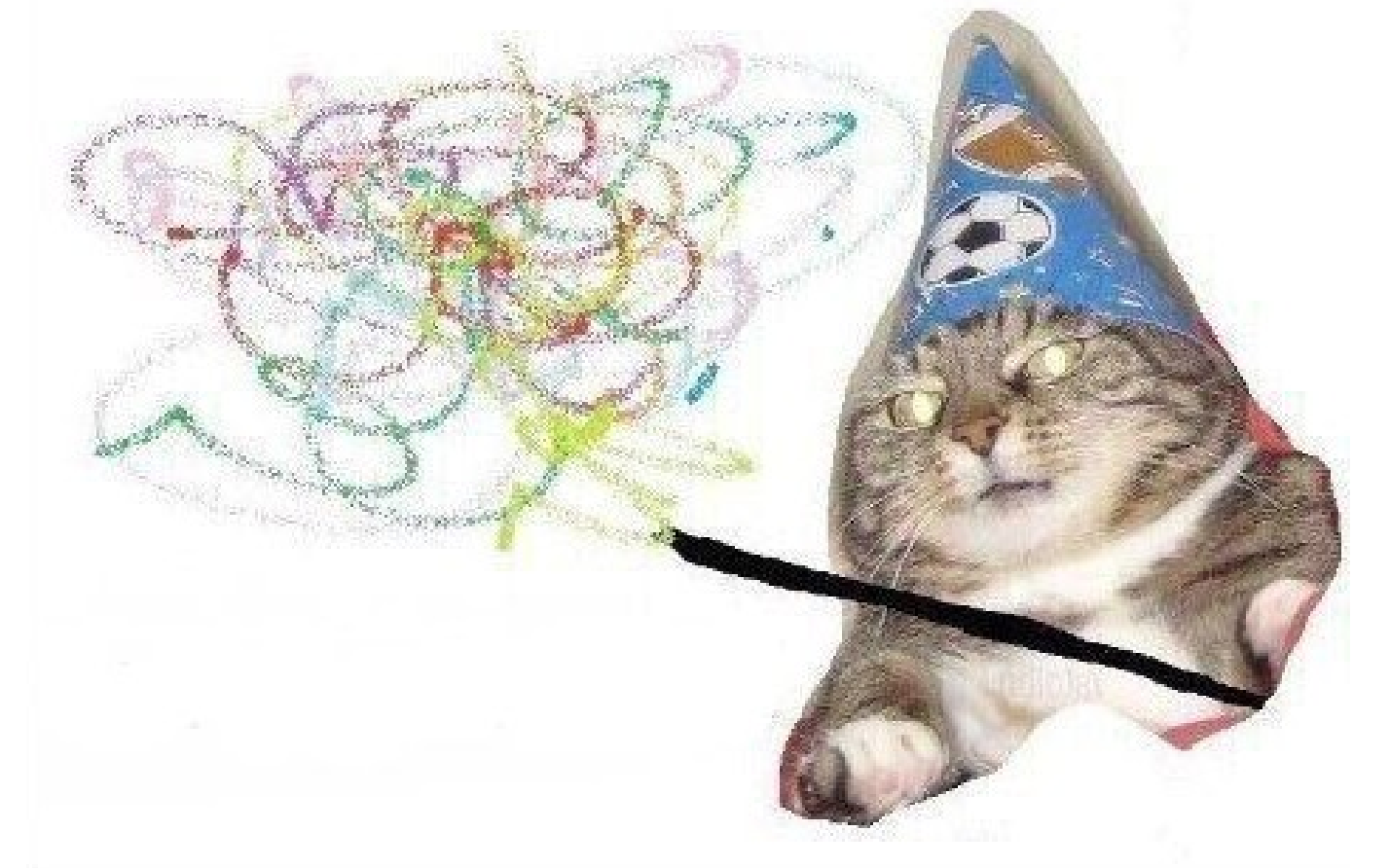
```
wstring s1 = L"Hello";  
wstring s2 = L"heLLo";  
if (s1 == s2)  
    std::wcout << s1 << " and " << s2 << " are equal\n";
```

Выведет: hello and hello are equal

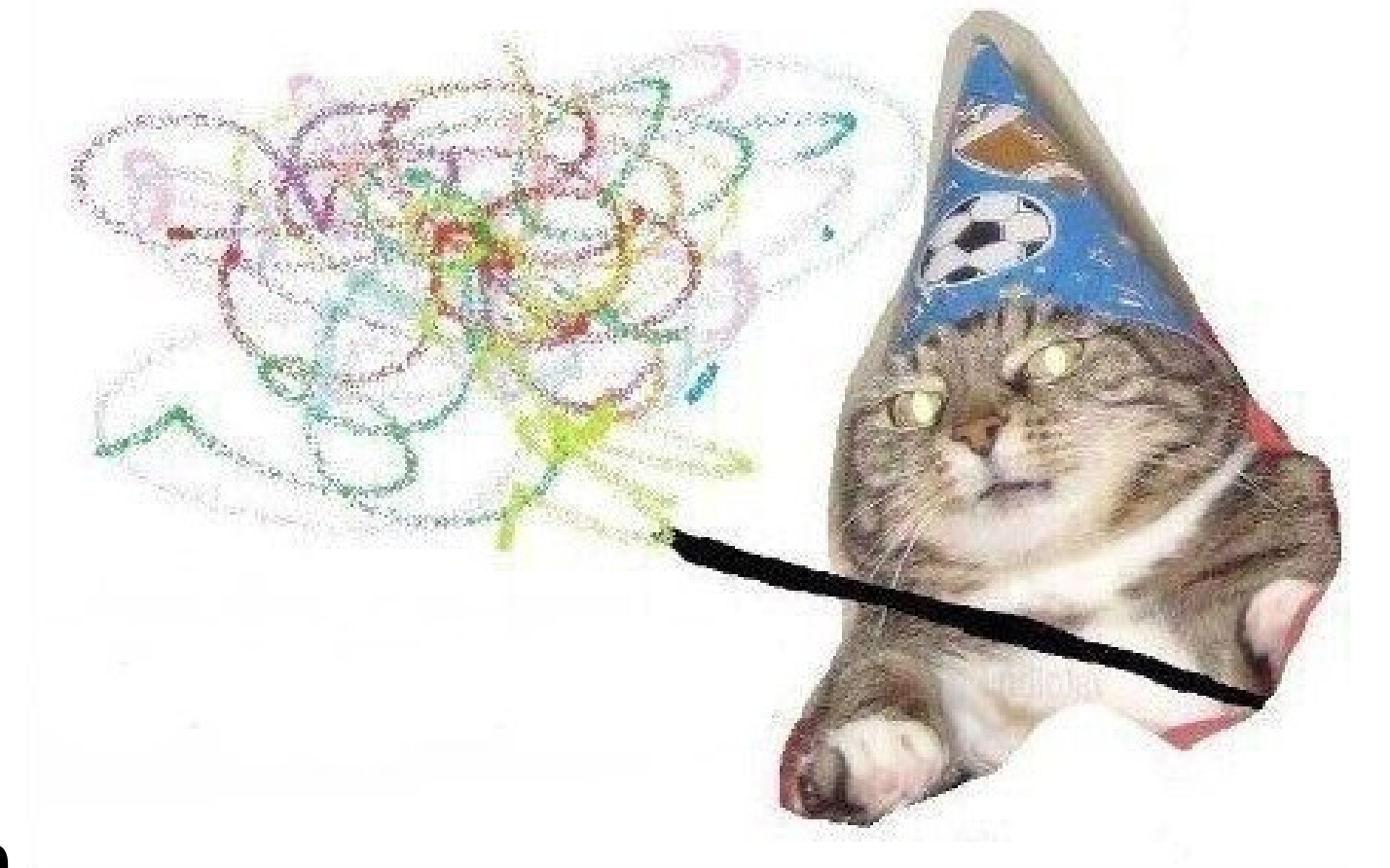
Сравнивать строки без учёта регистра

Сравнивать без учёта регистра

```
template <class Char>
struct ichar_traits : public std::char_traits<Char> {
    static bool eq(Char c1, Char c2) {
        return std::toupper(c1) == std::toupper(c2);
    }
    static bool lt(Char c1, Char c2) {
        return std::toupper(c1) < std::toupper(c2);
    }
    ...
};
```



Сравнивать без учёта регистра



...

```
static int compare(const Char* s1, const Char* s2, size_t n,   
    for (; n-- != 0; ++s1, ++s2)  
        if (std::toupper(*s1) < std::toupper(*s2)) return -1;  
        else if (std::toupper(*s1) > std::toupper(*s2)) return 1;  
    return 0;  
}
```

```
static const char* find(const Char* s, int n, Char a) {  
    for (a = std::toupper(a); n-- != 0; ++s)  
        if (std::toupper(*s) == a) return s;  
    return nullptr;  
}
```

Сравнивать без учёта регистра

```
template <class Char>  
using ibasic_string = std::basic_string<Char, ichar_traits<Char> >;  
using istring = ibasic_string<char>;  
using iwstring = ibasic_string<wchar_t>;
```

Сравнивать без учёта регистра

```
template <class Char>
using ibasic_string = std::basic_string<Char, ichar_traits<Char> >;
using istring = ibasic_string<char>;
using iwstring = ibasic_string<wchar_t>;

template <class Char, class Traits, class Char2>
std::basic_ostream<Char, Traits>& operator<<(
    std::basic_ostream<Char, Traits>& os, const ibasic_string<Char2>& str)
{
    return os.write(str.data(), str.size());
}
```

Сравнивать без учёта регистра

```
istring s1 = "Hello";  
istring s2 = "heLlo";  
if (s1 == s2)  
    std::cout << s1 << " and " << s2 << " are equal\n";
```

Выводит: Hello and heLlo are equal

Сравнивать без учёта регистра

```
wstring s1 = L"Hello";  
wstring s2 = L"heLLo";  
if (s1 == s2)  
    std::wcout << s1 << " and " << s2 << " are equal\n";
```

Выводит: Hello and heLLo are equal

Сравнивать без учёта регистра — бонус!

Сравнивать без учёта регистра — бонус!

```
template <class Char>
using ibasic_string_view = std::basic_string_view<Char, ichar_traits<Char> >;
using istring_view = ibasic_string_view<char>;
using iwstring_view = ibasic_string_view<wchar_t>;

template <class Char, class Traits, class Char2>
std::basic_ostream<Char, Traits>& operator<<(
    std::basic_ostream<Char, Traits>& os, const ibasic_string_view<Char2>& str)
{
    return os.write(str.data(), str.size());
}
```


Сравнивать без учёта регистра — бонус!

```
iwstring_view s1 = L"Hello";  
iwstring_view s2 = L"heLLo";  
if (s1 == s2)  
    std::wcout << s1 << " and " << s2 << " are equal\n";
```

Выводит: Hello and heLLo are equal

<algorithm>



Сделайте чтоб было быстро!

«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) $\Rightarrow O(N)$

for (size_t i = 0; i < N; ++i)
for (size_t j = 0; j < N; ++j) $\Rightarrow O(N^2)$

«O» большое

| N | $N \cdot \log(N)$ | $N \cdot N$ |
|-------|-------------------|-------------|
| 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

$\Rightarrow O(N \log_2(N))$

«O» большое

`std::sort` $\Rightarrow O(N \log_2(N))$

`std::stable_sort` $\Rightarrow O(N \log_2^2(N))$

«O» большое

`std::sort` $\Rightarrow O(N \log_2(N))$

~~`std::stable_sort`~~ $\Rightarrow O(N \log_2^2(N))$

`std::minmax_element` $\Rightarrow O(N)$

`std::nth_element` $\Rightarrow \sim O(N)$

`std::partial_sort` $\Rightarrow O(N \log_2(S))$

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 баллов

```
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 к медиане?



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



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);
```

$O(N \cdot \log(10))$

vs

$O(N \cdot \log(N))$

| N | $N \cdot \log(10)$ $N + (N/2 - 1) \cdot \log_2(9)$ | $N \cdot \log(N)$ | $N \cdot \log(N) - N \cdot \log(10)$ |
|------------|---|-------------------|--------------------------------------|
| 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 |

Гетерогенные сравнения



Компараторы

```
struct person;
```

```
bool operator<(const person& p, const std::string& name);
```

```
bool operator<(const std::string& name, const person& p);
```

Компараторы

```
struct person;
```

```
bool operator<(const person& p, const std::string& name);
```

```
bool operator<(const std::string& name, const person& p);
```

```
std::set<std::string> users = get_users();
```

```
person p = get_some_person();
```

```
std::cout << *users.find(p) << std::endl; // P.S.: .find(p) никогда не .end()
```

Компараторы

```
main.cpp:186:31: error: no matching function for call to
```

```
'std::set<std::basic_string<char> >::find(person&)'
```

```
std::cout << *users.find(p) << std::endl; //P.S.: .find(p) никогда не .end()
```

```
^
```

```
In file included from /usr/include/c++/6/set:61:0,
```

```
from ../experiments/dll4/urgent/corehard_sprint_2017/main.cpp:126:
```

```
/usr/include/c++/6/bits/stl_set.h:692:7: note: candidate: std::set<_Key, _Compare,
_Alloc>::iterator std::set<_Key, _Compare, _Alloc>::find(const key_type&) [with _Key =
std::basic_string<char>; _Compare = std::less<std::basic_string<char> >; _Alloc =
std::allocator<std::basic_string<char> >; std::set<_Key, _Compare, _Alloc>::iterator =
std::_Rb_tree_const_iterator<std::basic_string<char> >; std::set<_Key, _Compare,
_Alloc>::key_type = std::basic_string<char>]
```

`std::less<Something>` vs `std::less<>`

```
struct person;
```

```
bool operator<(const person& p, const std::string& name);
```

```
bool operator<(const std::string& name, const person& p);
```

```
std::set<std::string, std::less<>> users = get_users();
```

```
person p = get_some_person();
```

```
std::cout << *users.find(p) << std::endl; // P.S.: .find(p) никогда не .end()
```

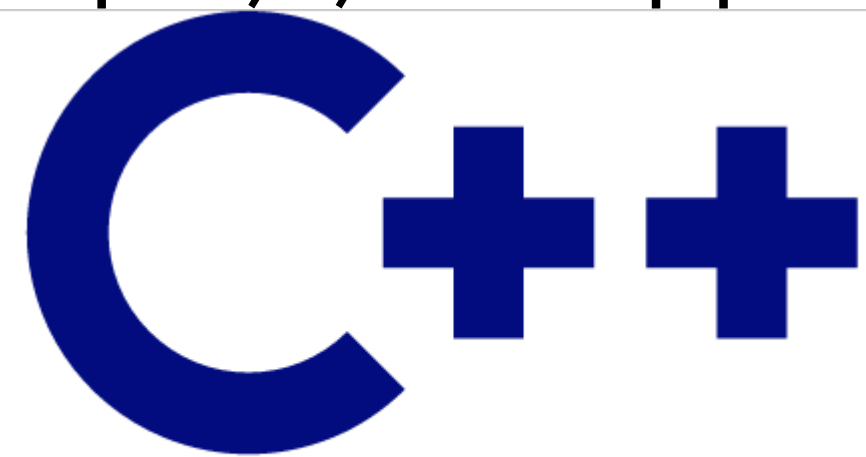
`std::less<Something>` vs `std::less<>`

```
std::cout << *users.find(p) << std::endl; // P.S.: .find(p) никогда не .end()
```

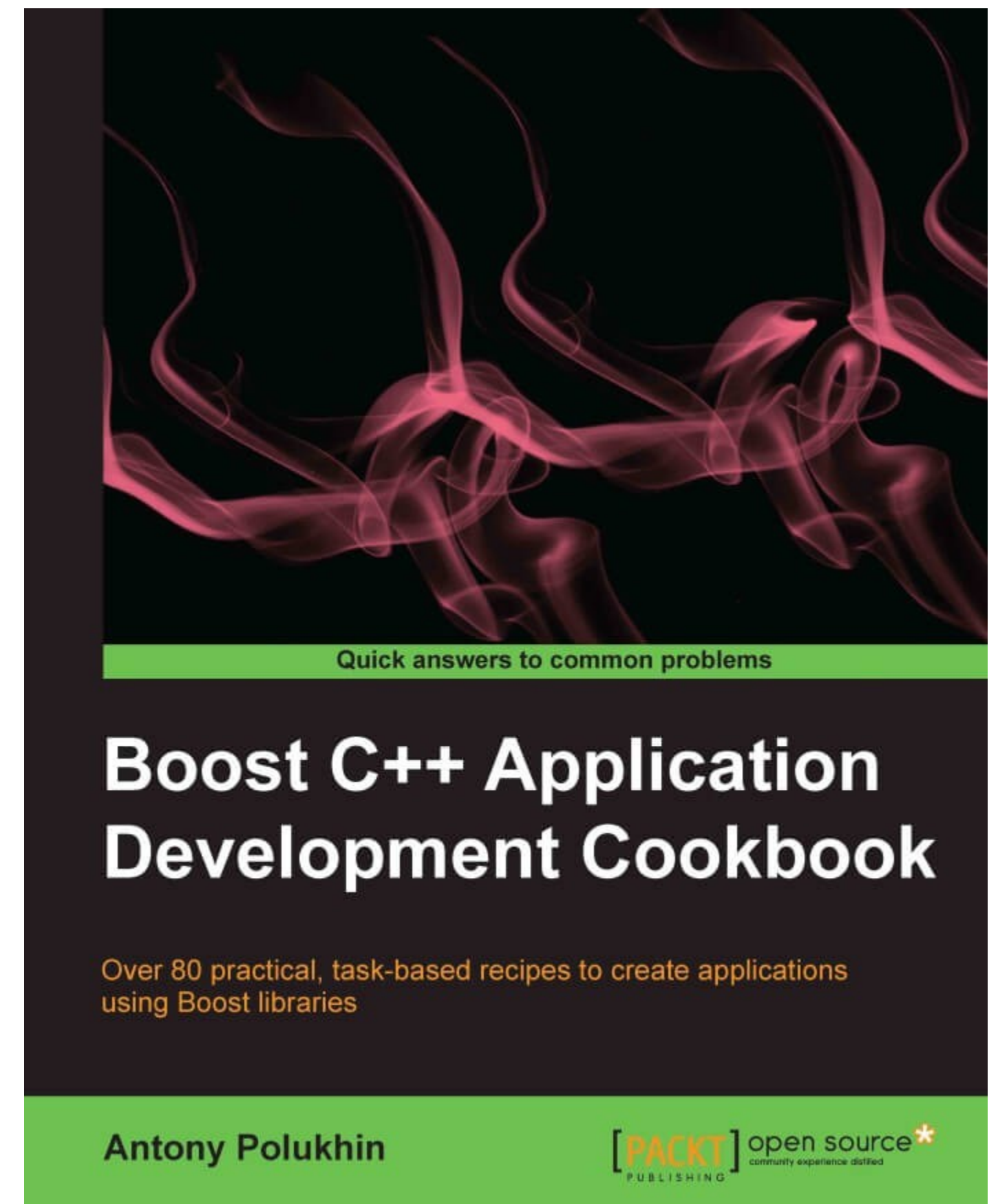
Вывод: Daenerys Targaryen, The Unburnt, Queen of the Andals, the Rhoynar and the First Men, Queen of Meereen, Khaleesi of the Great Grass Sea, Breaker of Chains, Mother of Dragons.

Спасибо! Вопросы?

<https://stdcpp.ru>



РГ21 C++ РОССИЯ



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