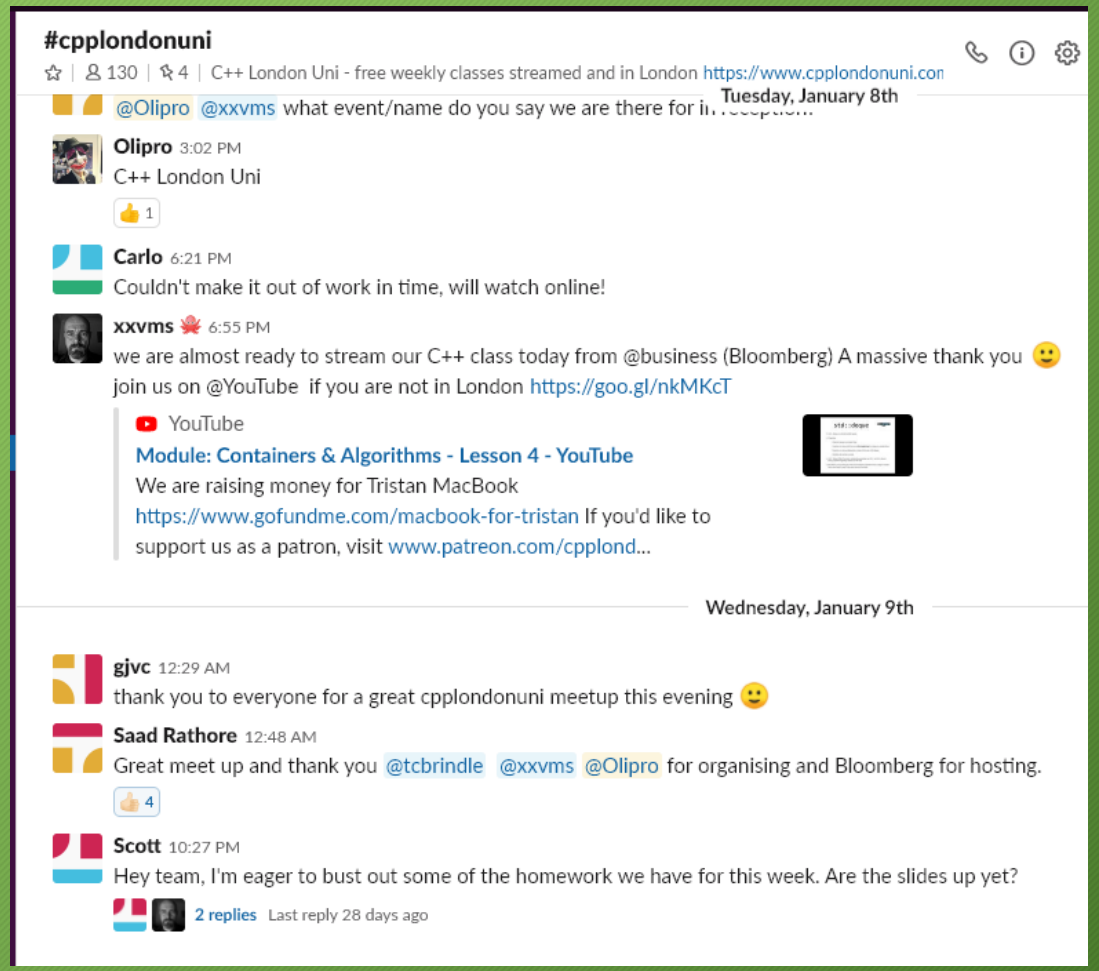
Getting to Know the Standard Library Session 4

Alex Voronov

Feedback

- We'd love to hear from you!
- The easiest way is via the *CPPLang* Slack organisation. Our chatroom is #cpplondonuni
- If you already use Slack, don't worry, it supports multiple workgroups!
- Go to https://slack.cpp.al to register.



Getting to Know the Standard Library

C++ London Uni

1. Introduction to unit testing with Catch2
2. Basic containers
   - std::vector
   - std::string
3. **Basics of the standard-library algorithms**
4. Associative containers
   - std::map and std::unordered_map
   - std::set and std::unordered_set
   - Associative containers with custom types
   - Set algorithms
5. More standard-library algorithms

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of an old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
- Summary
- Home exercise and additional content

We write unit tests instead of using std::cout

Catch2 is a single-header unit-testing framework

https://github.com/catchorg/Catch2

Download *catch.hpp* from the project page and include it in your .cpp file

```cpp
std::vector<std::string> split_into_words(const std::string &input) {
    std::vector<std::string> words;
    size_t word_begin = 0;

    while (true) {
        const size_t space_pos = input.find(' ', word_begin);
        const size_t word_length =
                space_pos == std::string::npos ? input.size() - word_begin
                                               : space_pos - word_begin;
        std::string new_word = input.substr(word_begin, word_length);
        if (!new_word.empty()) {
            words.push_back(new_word);
        }

        if (space_pos == std::string::npos) {
            break;
        } else {
            word_begin = space_pos + 1; // skip the space character
        }
    }

    return words;
}
```

Previous lesson exercise: Count substring occurrences

C++ London Uni

Write a function that given a string and a search substring returns the number of non-overlapping occurrences of the substring in the string

```
TEST_CASE("count occurrences") {
    CHECK(count_occurrences("banana", "ban") == 1);
    CHECK(count_occurrences("banana", "band") == 0);
    CHECK(count_occurrences("banana", "a") == 3);
    CHECK(count_occurrences("banana", "an") == 2);
    CHECK(count_occurrences("banana", "") == 0);

    CHECK(count_occurrences(std::string(8, 'a'), "a") == 8);
    CHECK(count_occurrences(std::string(8, 'a'), "aa") == 4);
    CHECK(count_occurrences(std::string(8, 'a'), "aaa") == 2);
    CHECK(count_occurrences(std::string(8, 'a'), std::string(8, 'a')) == 1);
    CHECK(count_occurrences(std::string(8, 'a'), std::string(9, 'a')) == 0);
    CHECK(count_occurrences("", "a") == 0);
}
```

Previous lesson exercise: Possible solution

C++ London Uni

```
size_t
count_occurrences(const std::string &string, const std::string &substring) {
    if (substring.empty()) {
        return 0u;
    }

    size_t count = 0u;
    size_t start_pos = 0u;
    while (true) {
        start_pos = string.find(substring, start_pos);
        if (start_pos == std::string::npos) {
            return count;
        }
        ++count;
        start_pos += substring.size();
    }
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- **Revision of old problem:**
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
- Summary
- Home exercise and additional content

Back to counting positive numbers

```cpp
size_t count_positive(const std::vector<int> &numbers) {
    size_t count = 0u;
    for (auto number : numbers) {
        if (number > 0) {
            ++count;
        }
    }

    return count;
}

TEST_CASE("Count positive numbers") {
    CHECK(count_positive({1, 2, 3, 4, 5}) == 5);
    CHECK(count_positive({1, 0, 0, 0, 1}) == 2);
    CHECK(count_positive({-1, -1, 0, 0, 42, 27}) == 2);
}
```

Introduce count_if with a predicate

```cpp
bool is_positive(int number) { return number > 0; }

size_t count_positive(const std::vector<int> &numbers) {
    return std::count_if(numbers.cbegin(), numbers.cend(),
            /*predicate*/ is_positive);
}


TEST_CASE("Count positive numbers") {
    CHECK(count_positive({1, 2, 3, 4, 5}) == 5);
    CHECK(count_positive({1, 0, 0, 0, 1}) == 2);
    CHECK(count_positive({-1, -1, 0, 0, 42, 27}) == 2);
}
```

Use a lambda function as a predicate

```cpp
size_t count_positive(const std::vector<int> &numbers) {
    auto is_positive = [](int number) { return number > 0; };
    return std::count_if(numbers.cbegin(), numbers.cend(),
            /*predicate*/ is_positive);
}

TEST_CASE("Count positive numbers") {
    CHECK(count_positive({1, 2, 3, 4, 5}) == 5);
    CHECK(count_positive({1, 0, 0, 0, 1}) == 2);
    CHECK(count_positive({-1, -1, 0, 0, 42, 27}) == 2);
}
```

Define the lambda function in place

```cpp
size_t count_positive(const std::vector<int> &numbers) {
    return std::count_if(numbers.cbegin(), numbers.cend(),
        /*predicate*/ [](int number) { return number > 0; });
}

TEST_CASE("Count positive numbers") {
    CHECK(count_positive({1, 2, 3, 4, 5}) == 5);
    CHECK(count_positive({1, 0, 0, 0, 1}) == 2);
    CHECK(count_positive({-1, -1, 0, 0, 42, 27}) == 2);
}
```

And don’t forget to #include <algorithm>

C++ London Uni

```cpp
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

#include <vector>
#include <algorithm>

size_t count_positive(const std::vector<int> &numbers) {
    return std::count_if(numbers.cbegin(), numbers.cend(),
        /*predicate*/ [](int number) { return number > 0; });
}

TEST_CASE("Count positive numbers") {
    CHECK(count_positive({1, 2, 3, 4, 5}) == 5);
    CHECK(count_positive({1, 0, 0, 0, 1}) == 2);
    CHECK(count_positive({-1, -1, 0, 0, 42, 27}) == 2);
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- **Why do we use algorithms + quick overview**
- Quick intro into lambda expressions
- New problem: Palindrome phrases
- Summary
- Home exercise and additional content

STL algorithms: Motivation to learn

C++ London Uni

- More expressive
  Most of the names describe what they do, which makes the program more readable
- More reliable
  Well tested. No need to worry about correct exit conditions or handling empty containers
- Faster
  Better in computational complexity than naïve implementations

STL algorithms: Motivation to learn

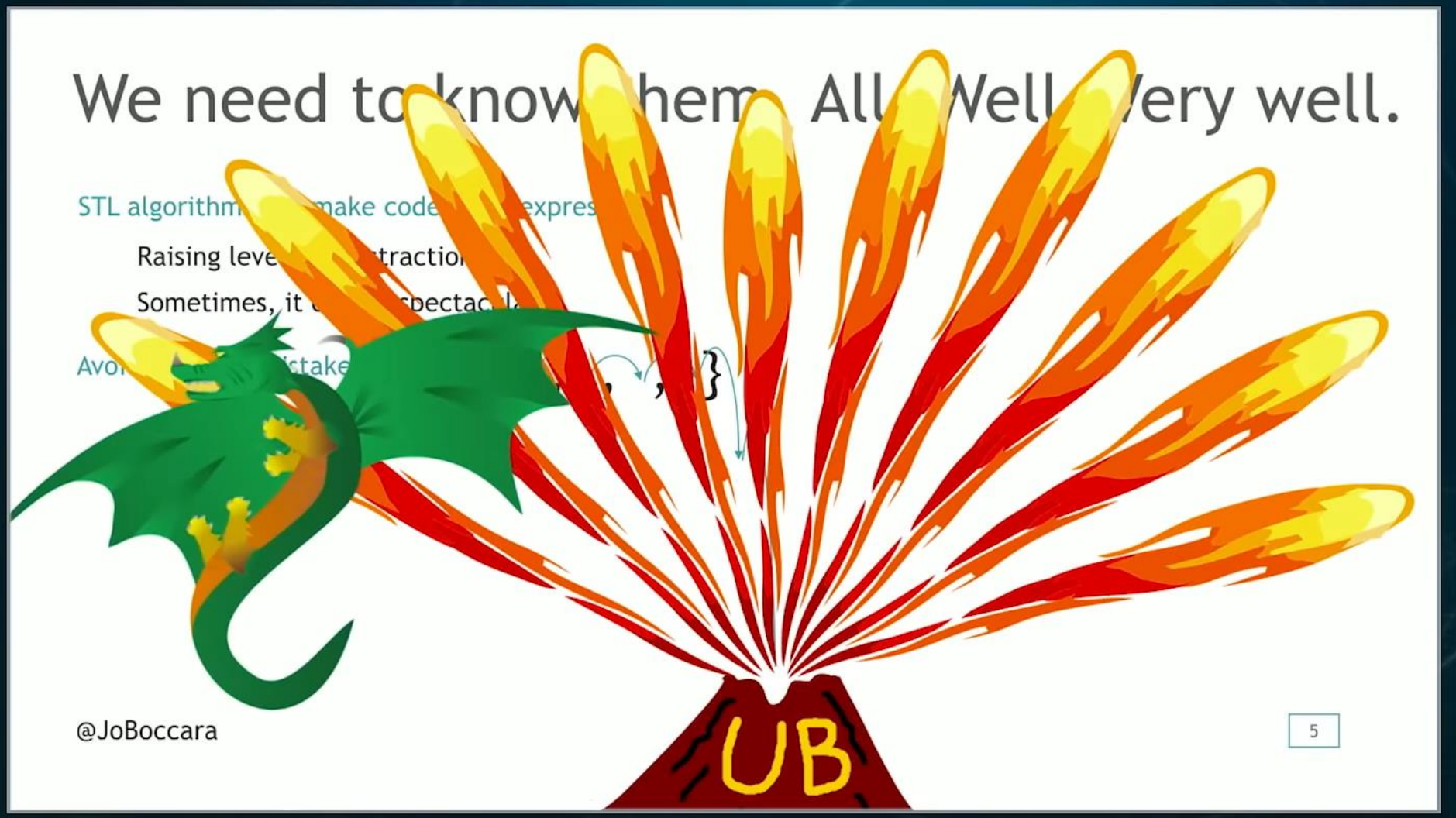
C++ London Uni

- More expressive
  Most of the names describe what they do, which makes the program more readable
- More reliable
  Well tested. No need to worry about correct exit conditions or handling empty containers
- Faster
  Better in computational complexity than naïve implementations

```
#include <algorithm>
#include <numeric>
```


Province of Value Queries
count
all_of
any_of
none_of
transform_inclusive_scan
Valley of Partial Sums
transform_exclusive_scan
inclusive_scan
exclusive_scan
partial_sum
Province of Property Queries
Province of 2-Ranges Properties
equal
mismatch
lexicographical_compare
is_permutation
set_difference
set_symmetric_difference
Glorious County of Algos on Sets
set_intersection
set_union
includes
merge
Lonely Islands
for_each
transform
uninitialized_value_construct
uninitialized_default_construct
uninitialized_copy
uninitialized_fill
uninitialized_move
destroy
copy_backward
Lands of Queries
find
adjacent_find
transform_reduce
Value Searchers
Lagoon of Folds
reduce
equal_range
upper_bound
inner_product
accumulate
binary_search
lower_bound
sample
adjacent_difference
Peninsula of Raw Memory
Territory of Movers
copy
move
swap_ranges
move_backward
Province of Research
max_element
min_element
Bay of Range Searches
find_end
Relative Value Searchers
minmax_element
find_first_of
search
remove
unique
Island of Structure Changers
iota
fill
generate
replace
Land of Value Modifiers
Lands of Permutations
partition_point
partition
sort
partial_sort
Shore of Sorting
nth_element
sort_heap
inplace_merge
reverse
rotate
make_heap
pop_heap
push_heap
next_permutation
prev_permutation
shuffle
Province of Heaps
Secret Runes
stable_*
*_copy
*_if
is_*_until
is_*
random_shuffle (wreck)
The World of C++ STL Algorithms
Fluent C++
Fluent C++
fluentcpp.com

C++ London Uni



105 STL Algorithms in Less Than an Hour

https://youtu.be/2olsGf6JIkU

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- **Quick intro into lambda expressions**
- New problem: Palindrome phrases
- Summary
- Home exercise and additional content

Lambda expressions

A way do define an anonymous function right at the places of use. They work great with algorithms!

Couple of tips:

- Prefer smaller lambda functions for better readability
- If you need to name them, use `auto` type specifier

Lambda expressions: Introduction

```
TEST_CASE("lambda expressions introduction") {
    auto lambda_example =
        [/* captures */](/* function arguments */) {/* function body */};
    lambda_example(); // invocation

    auto add = [](int left, int right) { return left + right; };
    CHECK(add(2, 3) == 5);
    CHECK(add(-70, 70) == 0);

    // lambda return type example
    auto wrap_into_char =
        [](int value) -> unsigned char { return value % 256; };

    CHECK(wrap_into_char(0) == 0);
    CHECK(wrap_into_char(10) == 10);
    CHECK(wrap_into_char(-10) == 246);
    CHECK(wrap_into_char(600) == 88);
}
```

```
TEST_CASE("lambda captures") {
    int number = 3;
    SECTION("capture by value") {
        auto capture_by_value = [number]() { return number * number; };
        number = 7;
        CHECK(capture_by_value() == 9);
    }

    SECTION("capture by reference") {
        auto capture_by_reference = [&number]() { return number * number; };
        number = 7;
        CHECK(capture_by_reference() == 49);
    }

    SECTION("modify capture by reference") {
        auto modify_capture = [&number] { number += 2; };
        modify_capture();
        CHECK(number == 5);
    }
}
```

Capture defaults

```
int a = 5, b = 7, c = 20;
auto default_by_value = [=] { return a + b + c; };
auto default_by_reference = [&] { return a + b + c; };
auto default_by_value_with_exceptions = [=, &a] { return a + b + c; };
auto default_by_reference_with_exceptions = [&, c] { return a + b + c; };
```

Lambda expressions: Practice

Write a function that takes a vector of integer numbers and a reference number and returns how many numbers in a vector are greater than the reference number.

Use std::count_if with a lambda expression

```
TEST_CASE("count greater than") {
    CHECK(count_greater_than({0, 0, 2, 2, 3}, 0) == 3);
    CHECK(count_greater_than({0, 0, 2, 2, 3}, 1) == 3);
    CHECK(count_greater_than({0, 0, 2, 2, 3}, 2) == 1);
}
```

Lambda expressions: Practice

C++ London Uni

```
size_t count_greater_than(const std::vector<int> &numbers, int reference) {
    return std::count_if(
        numbers.begin(), numbers.end(),
        [reference](const int entry) { return entry > reference; });
}

TEST_CASE("count greater than") {
    CHECK(count_greater_than({0, 0, 2, 2, 3}, 0) == 3);
    CHECK(count_greater_than({0, 0, 2, 2, 3}, 1) == 3);
    CHECK(count_greater_than({0, 0, 2, 2, 3}, 2) == 1);
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- **New problem: Palindrome phrases**
- Summary
- Home exercise and additional content

Next problem: Palindrome phrases

C++ London Uni

```
TEST_CASE("is palindrome") {
    CHECK(is_palindrome("Nurses run."));
    CHECK(is_palindrome("A Man, A Plan, A Canal: Panama!"));

    CHECK(is_palindrome("level"));
    CHECK(is_palindrome("Level"));
    CHECK(is_palindrome("Noon"));
    CHECK_FALSE(is_palindrome("Persimmon"));

    CHECK(is_palindrome(""));
    CHECK(is_palindrome(" "));
    CHECK(is_palindrome("!?"));
}
```

Palindrome phrases: Possible solution

```
bool is_palindrome(std::string phrase) {
    phrase.erase(
        std::remove_if(phrase.begin(), phrase.end(),
                       [](unsigned char c) { return !std::isalnum(c); }),
        phrase.end());

    const size_t half_length = phrase.size() / 2;
    return std::equal(
        phrase.cbegin() , phrase.cbegin()  + half_length,
        phrase.crbegin(), phrase.crbegin() + half_length,
        [](unsigned char left, unsigned char right) {
            return std::tolower(left) == std::tolower(right);
        });
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
  - **Find and count**
  - Text transformations
  - Remove-and-erase idiom
  - Checking iterator ranges for equality
- Summary
- Home exercise and additional content

std::count and std::count_if

C++ London Uni

```
TEST_CASE("count and count_if") {
    std::vector<std::string> fruit_basket{
        "banana", "apple", "orange", "banana", "persimmon"};

    CHECK(count(fruit_basket.begin(), fruit_basket.end(), "banana") == 2);
    CHECK(count_if(fruit_basket.begin(), fruit_basket.end(),
                   [](const std::string &fruit) { return fruit.size() >= 6; })
          == 4);
}
```

std::find

```
TEST_CASE("find") {
    std::vector<std::string> fruit_basket{
        "banana", "apple", "orange", "banana", "persimmon"};

    auto apple_it =
        std::find(fruit_basket.cbegin(), fruit_basket.cend(), "apple");
    REQUIRE(apple_it != fruit_basket.cend());
    CHECK(*apple_it == "apple");

    auto mango_it =
        std::find(fruit_basket.cbegin(), fruit_basket.cend(), "mango");
    REQUIRE(mango_it == fruit_basket.cend());

    // There is also find_if(...) that takes a predicate instead of a value.
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
  - Find and count
  - **Text transformations**
  - Remove-and-erase idiom
  - Checking iterator ranges for equality
- Summary
- Home exercise and additional content

Helpers: std::isalnum and std::tolower

```
TEST_CASE("isalnum") {
    CHECK(std::isalnum('a'));
    CHECK(std::isalnum('A'));
    CHECK(std::isalnum('Z'));
    CHECK(std::isalnum('7'));

    CHECK_FALSE(std::isalnum(' '));
    CHECK_FALSE(std::isalnum('-'));
    CHECK_FALSE(std::isalnum('.'));
}
```

```
TEST_CASE("tolower") {
    CHECK(std::tolower('a') == 'a');
    CHECK(std::tolower('A') == 'a');
    CHECK(std::tolower('Z') == 'z');
    CHECK(std::tolower('7') == '7');

    CHECK(std::tolower(' ') == ' ');
    CHECK(std::tolower('-') == '-');
    CHECK(std::tolower('.') == '.');
}
```

More from <cctype>:

- isalnum, isalpha, isdigit, isxdigit, ispunct, isspace
- islower, isupper, tolower, toupper
- isgraph, iscntrl, isblank, isprint

std::transform

```
TEST_CASE("transform: output is same as input") {
    std::string mixed_case{"MiXeD cAsE"};
    auto char_tolower = [](unsigned char c) { return std::tolower(c); };
    std::transform(mixed_case.cbegin(), mixed_case.cend(), mixed_case.begin(),
                   char_tolower);

    CHECK(mixed_case == "mixed case");
}
```

```cpp
TEST_CASE("transform: output is different from input") {
    const std::string mixed_case{"MiXeD cAsE"};
    auto char_tolower = [](unsigned char c) { return std::tolower(c); };

    std::string preallocated_string(mixed_case.size(), ' ');
    std::transform(mixed_case.cbegin(), mixed_case.cend(),
                   preallocated_string.begin(), char_tolower);
    CHECK(preallocated_string == "mixed case");

    std::string allocated_during_transform;
    // std::transform(mixed_case.cbegin(), mixed_case.cend(),
    //                allocated_during_transform, char_tolower);
    // would be undefined behaviour due to trying to write to an empty string

    std::transform(mixed_case.cbegin(), mixed_case.cend(),
                   std::back_inserter(allocated_during_transform),
                   char_tolower);
    CHECK(allocated_during_transform == "mixed case");
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
  - Find and count
  - Text transformations
  - **Remove-and-erase idiom**
  - Checking iterator ranges for equality
- Summary
- Home exercise and additional content

Back to erasing persimmons

C++ London Uni

```
std::vector<std::string> erase_persimmon(std::vector<std::string> fruit) {
    for (auto it = fruit.begin(); it != fruit.end();) {
        if (*it == "persimmon") {
            it = fruit.erase(it);
        } else {
            ++it;
        }
    }
    return fruit;
}

TEST_CASE("erase persimmon") {
    std::vector<std::string> fruit_basket{
        "banana", "orange", "persimmon", "apple", "persimmon"};
    CHECK(erase_persimmon(fruit_basket) ==
          std::vector<std::string>{"banana", "orange", "apple"});
}
```

Remove-and-erase idiom

```cpp
std::vector<std::string> erase_persimmon(std::vector<std::string> fruit) {
    // Drop persimmons and group remaining fruit at the head of the container
    auto begin_of_rubbish =
        std::remove(fruit.begin(), fruit.end(), "persimmon");
    // Trim the tail of the container that doesn't have any good fruit
    fruit.erase(/*from*/ begin_of_rubbish, /*to*/ fruit.end());
    return fruit;
}



TEST_CASE("erase persimmon") {
    std::vector<std::string> fruit_basket{
        "banana", "orange", "persimmon", "apple", "persimmon"};
    CHECK(erase_persimmon(fruit_basket) ==
          std::vector<std::string>{"banana", "orange", "apple"});
}
```

Remove-and-erase idiom: one line

```
std::vector<std::string> erase_persimmon(std::vector<std::string> fruit) {
    fruit.erase(
        std::remove(fruit.begin(), fruit.end(), "persimmon"), fruit.end());
    return fruit;
}
```

```
TEST_CASE("erase persimmon") {
    std::vector<std::string> fruit_basket{
        "banana", "orange", "persimmon", "apple", "persimmon"};
    CHECK(erase_persimmon(fruit_basket) ==
          std::vector<std::string>{"banana", "orange", "apple"});
}
```

Introduction to standard-library algorithms

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
  - Find and count
  - Text transformations
  - Remove-and-erase idiom
  - **Checking iterator ranges for equality**
- Summary
- Home exercise and additional content

std::equal

C++ London Uni

```
TEST_CASE("equal") {
    const std::string lowercase_word{"word"};
    const std::string capitalised_word{"Word"};

    CHECK_FALSE(std::equal(lowercase_word.cbegin(), lowercase_word.cend(),
                           capitalised_word.cbegin(), capitalised_word.cend()));
    CHECK(std::equal(lowercase_word.cbegin(), lowercase_word.cend(),
                     capitalised_word.cbegin(), capitalised_word.cend(),
                     [](unsigned char left, unsigned char right) {
                         return std::tolower(left) == std::tolower(right);
                     }));
}
```

std::equal: any predicate makes sense

```
TEST_CASE("equal: predicates and size check") {
    const std::string bear{"bear"};
    const std::string duck{"duck"};
    const std::string long_cat{"looooooong cat"};

    CHECK_FALSE(
        std::equal(bear.cbegin(), bear.cend(), duck.cbegin(), duck.cend()));

    auto always_equal = [](unsigned char, unsigned char) { return true; };
    CHECK(std::equal(
        bear.cbegin(), bear.cend(), duck.cbegin(), duck.cend(), always_equal));
    CHECK_FALSE(std::equal(
        bear.cbegin(), bear.cend(), long_cat.cbegin(), long_cat.cend(),
        always_equal));
}
```

std::equal: beware of 3-iterator form

C++ London Uni

```
TEST_CASE("equal without size check") {
    const std::string duck{"duck"};
    const std::string duckling{"duckling"};

    // 4 iterators: will check that sizes match
    CHECK_FALSE(std::equal(duck.cbegin(), duck.cend(),
                           duckling.cbegin(), duckling.cend()));
    // 3 iterators: no size check
    CHECK(std::equal(duck.cbegin(), duck.cend(), duckling.cbegin()));

    // std::equal(duckling.cbegin(), duckling.cend(), duck.cbegin())
    // is undefined behaviour because comparison will go past duck.cend()
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- **New problem: Palindrome phrases**
  - Find and count
  - Text transformations
  - Remove-and-erase idiom
  - Checking iterator ranges for equality
- Summary
- Home exercise and additional content

Revision of palindrome phrases solution

C++ London Uni

```
bool is_palindrome(std::string phrase) {
    phrase.erase(
        std::remove_if(phrase.begin(), phrase.end(),
                       [](unsigned char c) { return !std::isalnum(c); }),
        phrase.end());

    const size_t half_length = phrase.size() / 2;
    return std::equal(
        phrase.cbegin() , phrase.cbegin()  + half_length,
        phrase.crbegin(), phrase.crbegin() + half_length,
        [](unsigned char left, unsigned char right) {
            return std::tolower(left) == std::tolower(right);
        });
}
```

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
  - Find and count
  - Text transformations
  - Remove-and-erase idiom
  - Checking iterator ranges for equality
- **Summary**
- Home exercise and additional content

Summary

- Prefer algorithms over raw loops to make your code more expressive, more reliable and faster
- Lambda expressions work great with STL algorithms. They allow to declare a small function for comparison on transformation right in place of the use
- There are many algorithms, but the principles of how to use them are similar

Each time when writing a loop stop and think:
Is there an algorithm for this?

Introduction to standard-library algorithms

C++ London Uni

- Recap and updates
- Revision of old problem:
  Counting positive numbers with an algorithm
- Why do we use algorithms + quick overview
- Quick intro into lambda expressions
- New problem: Palindrome phrases
  - Find and count
  - Text transformations
  - Remove-and-erase idiom
  - Checking iterator ranges for equality
- Summary
- **Home exercise and additional content**

Home exercise: Anagrams

Write a function that given two strings tells that one is an anagram of the other taking into account only letters and digits

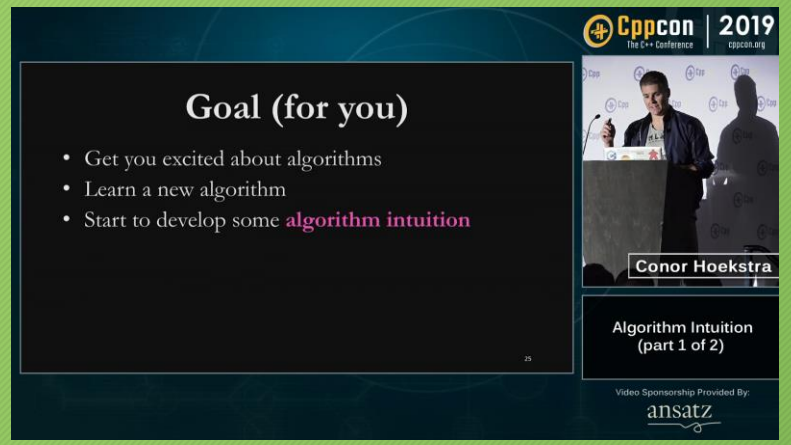
```
TEST_CASE("is anagram") {
    CHECK(is_anagram("Tom Marvolo Riddle", "I am Lord Voldemort"));
    CHECK(is_anagram("The Great Britain", "Tea? Bring it, heart!"));
    CHECK_FALSE(is_anagram("Tom Marvolo Riddle", "The Great Britain"));
}
```

More on STL algorithms

Jonathan Boccara

105 STL Algorithms in Less Than an Hour

https://www.youtube.com/watch?v=2olsGf6JIkU



Conor Hoekstra

Algorithm Intuition

https://www.youtube.com/watch?v=pUEnO6SvAMo

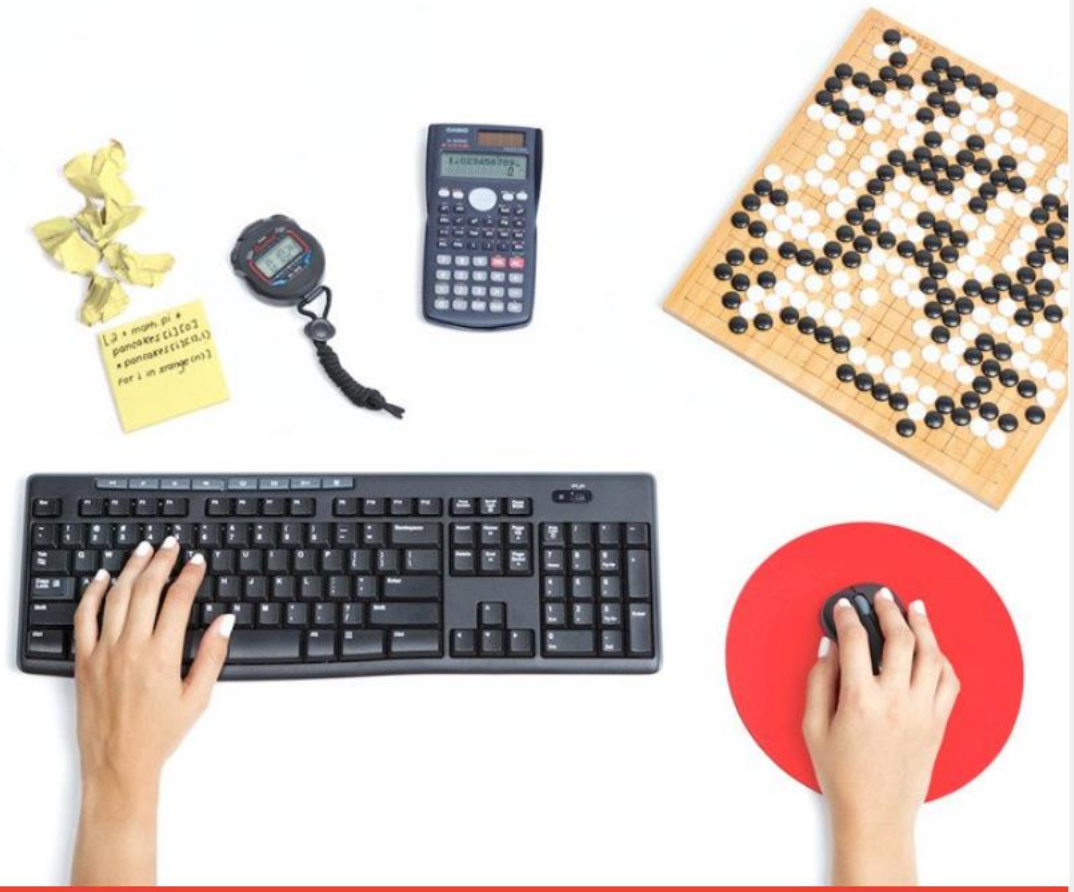
code jam About Schedule FAQ Rules Archive

Qualification Round on Saturday April, 4

Registration for Code Jam 2020 is open

Code Jam is back for its 17th year! Join the Code Jam community and take on a series of challenging algorithmic puzzles designed by Google engineers. You'll have a chance to earn the coveted title of Code Jam Champion and win $15,000 USD at the World Finals. Do you have what it takes?

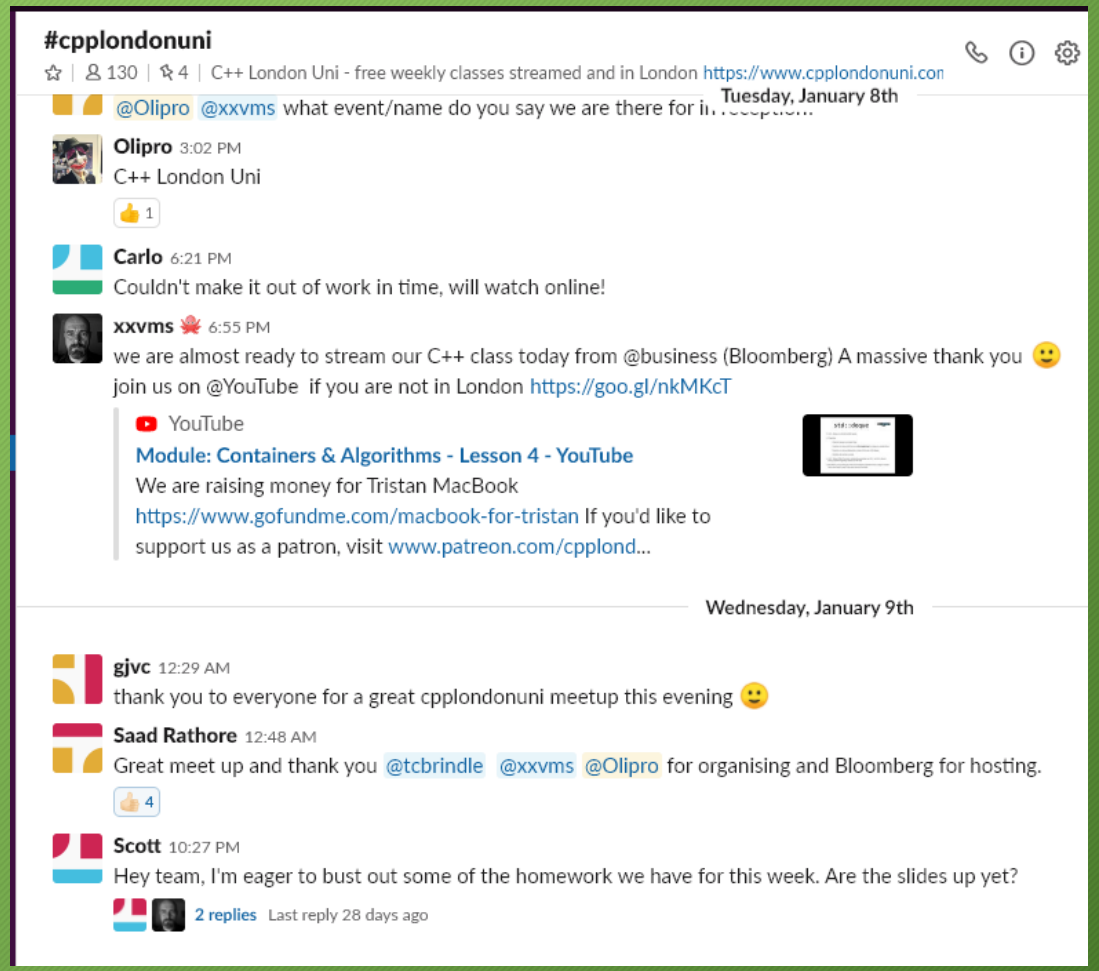
Register now



code jam

Feedback

- We'd love to hear from you!
- The easiest way is via the *CPPLang* Slack organisation. Our chatroom is #cpplondonuni
- If you already use Slack, don't worry, it supports multiple workgroups!
- Go to https://slack.cpp.al to register.



Thank You!

As usual, we will be going to the pub! Support us @ https://patreon.com/CPPLondonUni