Getting to Know the Standard Library Session 3

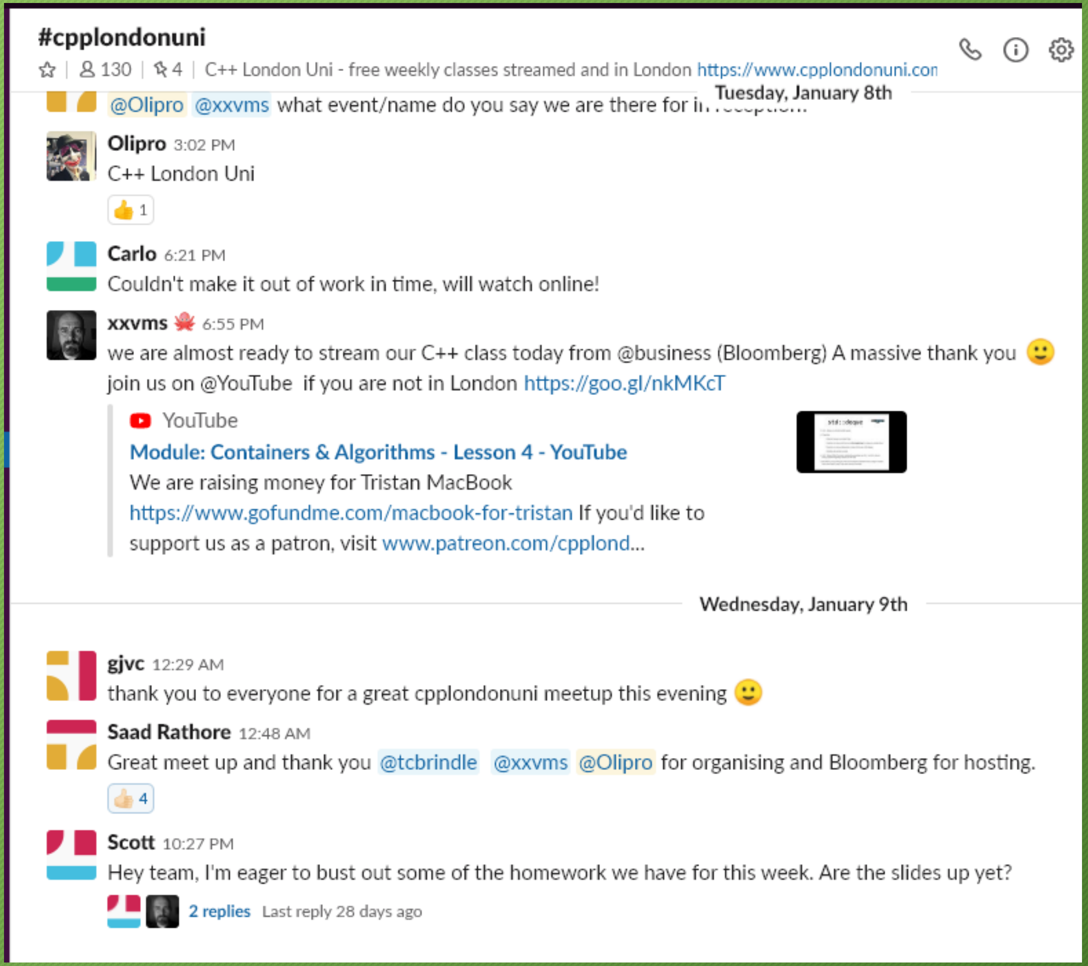
Alex Voronov

Getting to Know the Standard Library

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

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.



#cpplondonuni

☆ | 130 | 4 | C++ London Uni - free weekly classes streamed and in London https://www.cpplondonuni.con

Tuesday, January 8th

@Olipro @xxvms what event/name do you say we are there for in reception...

Olipro 3:02 PM
C++ London Uni
👍 1

Carlo 6:21 PM
Couldn't make it out of work in time, will watch online!

xxvms 6:55 PM
we are almost ready to stream our C++ class today from @business (Bloomberg) A massive thank you 🙂
join us on @YouTube if you are not in London https://goo.gl/nkMKcT

YouTube
Module: Containers & Algorithms - Lesson 4 - YouTube
We are raising money for Tristan MacBook https://www.gofundme.com/macbook-for-tristan If you'd like to support us as a patron, visit www.patreon.com/cpplond...

Wednesday, January 9th

gjvc 12:29 AM
thank you to everyone for a great cpplondonuni meetup this evening 🙂

Saad Rathore 12:48 AM
Great meet up and thank you @tcbrindle @xxvms @Olipro for organising and Bloomberg for hosting.
👍 4

Scott 10:27 PM
Hey team, I'm eager to bust out some of the homework we have for this week. Are the slides up yet?
2 replies Last reply 28 days ago

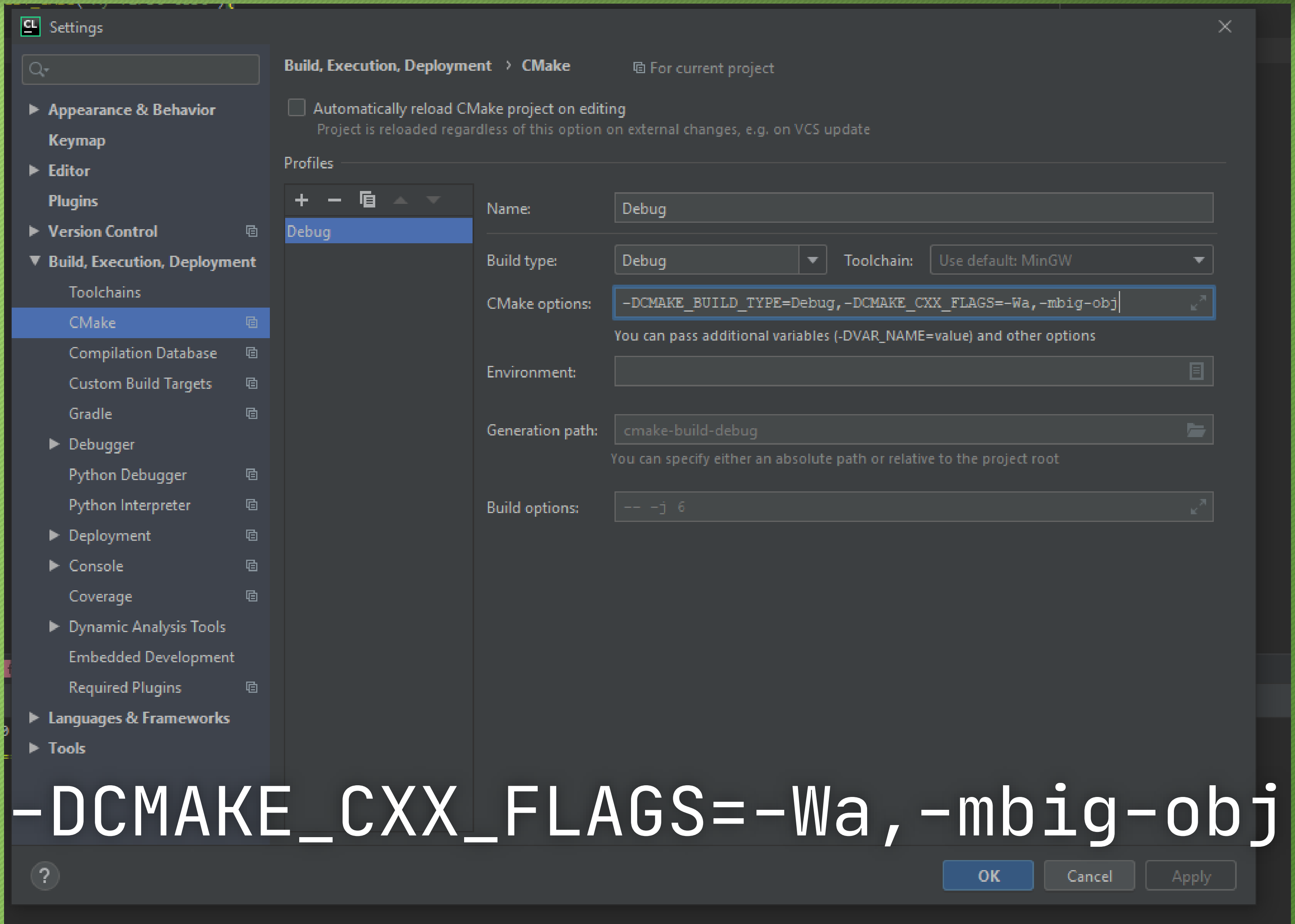
std::string: Session plan

- Recap and updates
- Introduction
- Construction and composition
- Substring: creation and lookup
- Conversions between strings and numbers
- Practice
- Summary
- Home exercise

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

#include <vector>

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);
}
```

```cpp
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"});
}
```

Update: fix for build with CLion on Windows



Settings

Build, Execution, Deployment › CMake   For current project

☐ Automatically reload CMake project on editing
Project is reloaded regardless of this option on external changes, e.g. on VCS update

Profiles

Debug

Name: Debug

Build type: Debug   Toolchain: Use default: MinGW

CMake options: `-DCMAKE_BUILD_TYPE=Debug,-DCMAKE_CXX_FLAGS=-Wa,-mbig-obj`

You can pass additional variables (-DVAR_NAME=value) and other options

Environment:

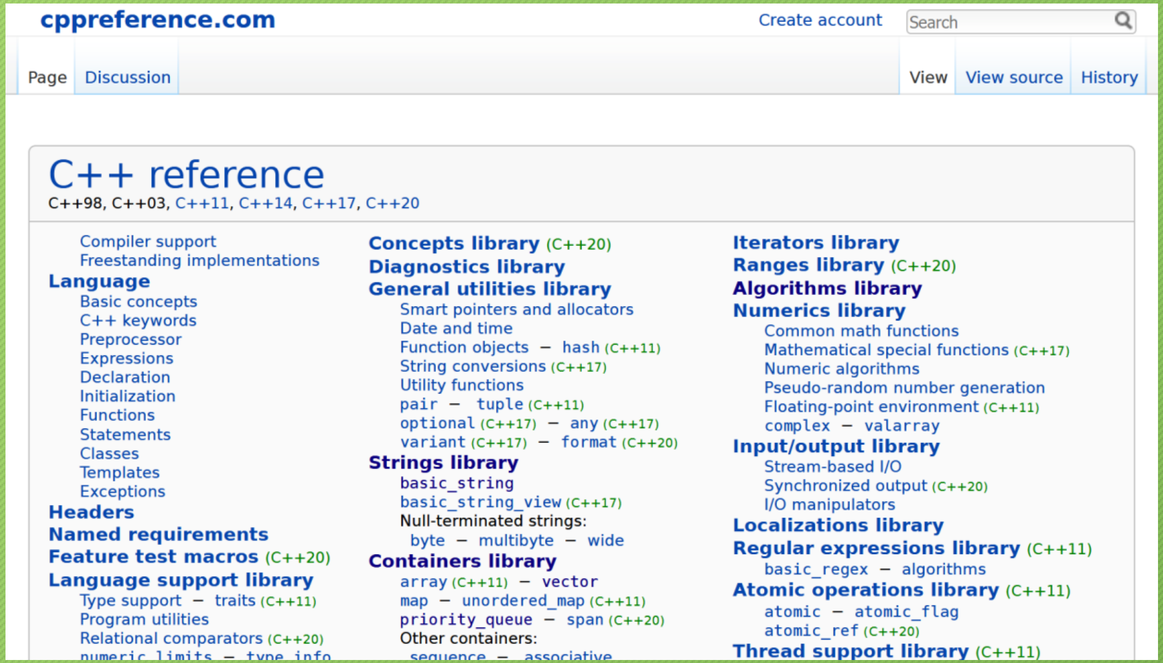
Generation path: cmake-build-debug

You can specify either an absolute path or relative to the project root

Build options: `-- -j 6`

```
-DCMAKE_CXX_FLAGS=-Wa,-mbig-obj
```

Update: C++ reference websites



cppreference.com Create account Search

Page Discussion View View source History

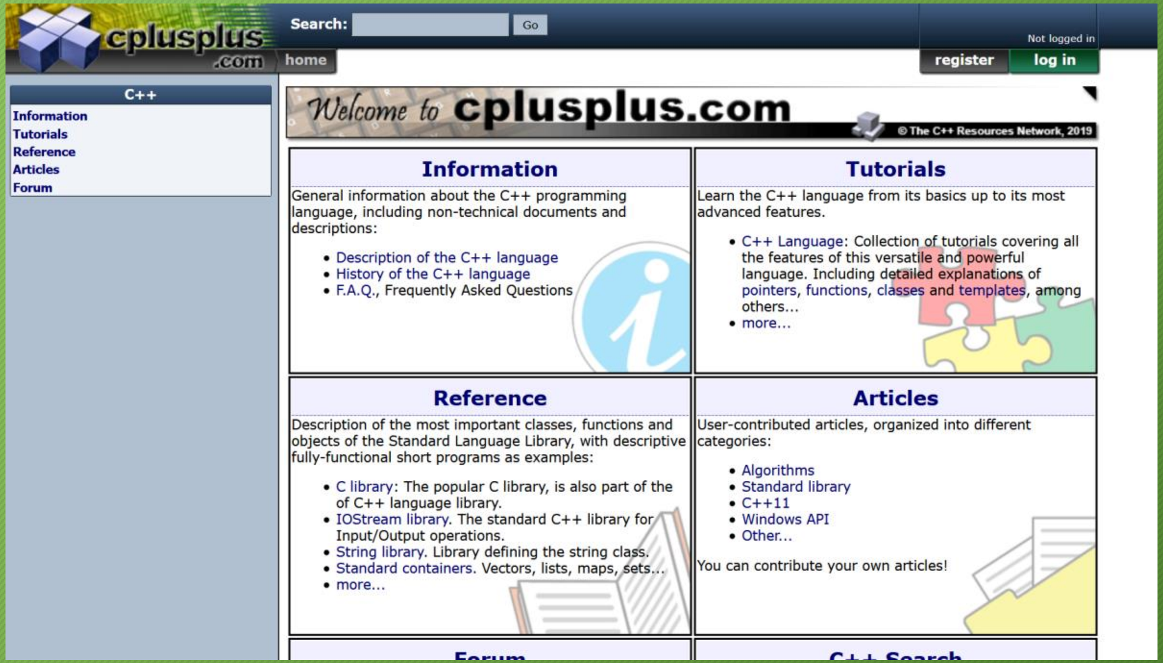
C++ reference

C++98, C++03, C++11, C++14, C++17, C++20

Compiler support
Freestanding implementations
Language
Basic concepts
C++ keywords
Preprocessor
Expressions
Declaration
Initialization
Functions
Statements
Classes
Templates
Exceptions
Headers
Named requirements
Feature test macros (C++20)
Language support library
Type support — traits (C++11)
Program utilities
Relational comparators (C++20)
numeric_limits — type_info

Concepts library (C++20)
Diagnostics library
General utilities library
Smart pointers and allocators
Date and time
Function objects — hash (C++11)
String conversions (C++17)
Utility functions
pair — tuple (C++11)
optional (C++17) — any (C++17)
variant (C++17) — format (C++20)
Strings library
basic_string
basic_string_view (C++17)
Null-terminated strings:
byte — multibyte — wide
Containers library
array (C++11) — vector
map — unordered_map (C++11)
priority_queue — span (C++20)
Other containers:
sequence — associative

Iterators library
Ranges library (C++20)
Algorithms library
Numerics library
Common math functions
Mathematical special functions (C++17)
Numeric algorithms
Pseudo-random number generation
Floating-point environment (C++11)
complex — valarray
Input/output library
Stream-based I/O
Synchronized output (C++20)
I/O manipulators
Localizations library
Regular expressions library (C++11)
basic_regex — algorithms
Atomic operations library (C++11)
atomic — atomic_flag
atomic_ref (C++20)
Thread support library (C++11)



cplusplus.com Search: Go Not logged in

home register log in

C++
Information
Tutorials
Reference
Articles
Forum

Welcome to cplusplus.com
© The C++ Resources Network, 2019

Information
General information about the C++ programming language, including non-technical documents and descriptions:
- Description of the C++ language
- History of the C++ language
- F.A.Q., Frequently Asked Questions

Tutorials
Learn the C++ language from its basics up to its most advanced features.
- C++ Language: Collection of tutorials covering all the features of this versatile and powerful language. Including detailed explanations of pointers, functions, classes and templates, among others...
- more...

Reference
Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples:
- C library: The popular C library, is also part of the of C++ language library.
- IOStream library. The standard C++ library for Input/Output operations.
- String library. Library defining the string class.
- Standard containers. Vectors, lists, maps, sets...
- more...

Articles
User-contributed articles, organized into different categories:
- Algorithms
- Standard library
- C++11
- Windows API
- Other...

You can contribute your own articles!

Forum C++ Search

cppreference.com cplusplus.com

std::string: Session plan

- Recap and updates
- **Introduction**
- Construction and composition
- Substrings: creation and lookup
- Conversion between strings and numbers
- Practice
- Summary
- Home exercise

Disclaimer about Unicode

Problem: Split a string into words

Write a function that
for a string of space-separated words
returns a vector of these words

```
TEST_CASE("split into words") {
    CHECK(split_into_words("banana apple potato") ==
          std::vector<std::string>{"banana", "apple", "potato"});
}
```

```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;
}
```

std::string

A container for efficient operations with strings

- Similar to std::vector of char
- In addition supports
    - Conversion from and to C-style strings
    - Substring or character lookup with `find()`, `rfind()`, `find_first_of()`, `find_last_of()` and so on
    - Substring extraction with `substr()`
    - Conversion from numbers to their string representation and vice versa
    - Composition with `.append()` or `operator+`

std::string: Session plan

- Recap and updates
- Introduction
- **Construction and composition**
- Substrings: creation and lookup
- Conversion between strings and numbers
- Practice
- Summary
- Home exercise

String construction

```cpp
TEST_CASE("string construction") {
    const std::string empty_string;
    CHECK(empty_string.empty());
    CHECK(empty_string == "");

    CHECK(std::string(4, 'z') == "zzzz");

    const std::string characters{"characters"};
    const std::string copy_of_characters = characters;
    CHECK(characters == copy_of_characters);

    CHECK(std::string{characters.begin() + 4, characters.begin() + 7} == "act");
    CHECK(std::string{characters.rbegin(), characters.rend()} == "sretcarahc");
}
```

```
TEST_CASE("string composition") {
    std::string example{"app"};

    SECTION("append") {
        example.append("end");
        CHECK(example == "append");

        CHECK(std::string{"crafts"}.append("man").append("ship") ==
              "craftsmanship");
    }

    SECTION("operator +=") {
        example += "end";
        CHECK(example == "append");
    }

    SECTION("operator +") {
        std::string appended = example + "end";
        CHECK(example == "app");
        CHECK(appended == "append");
    }
}
```

std::string: Session plan

- Recap and updates
- Introduction
- Construction and composition
- **Substrings: creation and lookup**
- Conversion between strings and numbers
- Practice
- Summary
- Home exercise

Getting a substring

```
TEST_CASE("substring") {
    const std::string abandon{"abandon"};

    CHECK(abandon.substr() == "abandon");
    CHECK(abandon.substr(/*start position*/ 4) == "don");
    CHECK(abandon.substr(/*start position*/ 4, /*count*/ 2) == "do");

    CHECK(abandon.substr(/*start position*/ 4, /*count*/ 100500) == "don");
    CHECK(abandon.substr(/*start position*/ 7) == "");

    CHECK_THROWS_AS(abandon.substr(/*start position*/ 8), std::out_of_range);
}
```

Find a character or a substring

```
TEST_CASE("string find and reverse find") {
    const std::string baden_baden{"Baden-Baden"};

    CHECK(baden_baden.find('B') == 0);
    CHECK(baden_baden.find('a') == 1);
    CHECK(baden_baden.find('a', /*start position*/ 2) == 7);
    CHECK(baden_baden.find('z') == std::string::npos);

    CHECK(baden_baden.find("ad") == 1);
    CHECK(baden_baden.find("ad", /*start position*/ 1) == 1);
    CHECK(baden_baden.find("ad", /*start position*/ 2) == 7);
    CHECK(baden_baden.find("ad", /*start position*/ 8) == std::string::npos);
    CHECK(baden_baden.find("lad") == std::string::npos);

    CHECK(baden_baden.rfind("ad") == 7);
}
```

Look up any character of a group

```
TEST_CASE("find a character from a group") {
    const std::string baden_baden{"Baden-Baden"};

    CHECK(baden_baden.find_first_of("nd") == 2);
    CHECK(baden_baden.find_last_of("nd") == 10);
    CHECK(baden_baden.find_first_not_of("nd") == 0);
    CHECK(baden_baden.find_last_not_of("nd") == 9);

    CHECK(baden_baden.find_first_of("nd", /*start position*/ 3) == 4);
}
```

std::string: Session plan

- Recap and updates
- Introduction
- Construction and composition
- Substrings: creation and lookup
- **Conversion between strings and numbers**
- Practice
- Summary
- Home exercise

String to number

```cpp
TEST_CASE("string to number") {
    CHECK(std::stoi("42") == 42);
    CHECK(std::stoi("-27") == -27);
    CHECK(std::stoi("6 lessons") == 6);
    CHECK(std::stoi("3.1415926") == 3);
    CHECK(std::stof("3.1415926") == Approx(3.1415926f));

    CHECK(std::stoi("1101", nullptr, /*number base*/ 2) == 13);

    CHECK_THROWS_AS(std::stoi("lesson 2"), std::invalid_argument);

    CHECK(std::stoi("2'000'000'000") == 2);
    CHECK(std::stoi("2 000 000 000") == 2);
    CHECK(std::stoi("2000000000") == 2000000000);
    CHECK_THROWS_AS(std::stoi("3000000000"), std::out_of_range);
}
```

String to number: types

- `stoi` → integer
- `stol` → long
- `stoll` → long long
- `stoul` → unsigned long
- `stoull` → unsigned long long
- `stof` → float
- `stod` → double
- `stold` → long double

Number to string

```
TEST_CASE("number to string") {
    CHECK(std::to_string(42) == "42");
    CHECK(std::to_string(-27) == "-27");
    CHECK(std::to_string(3.1415926f) == "3.141593");
    CHECK(std::to_string(1e-5) == "0.000010");
}
```

```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;
}
```

std::string: Session plan

- Recap and updates
- Introduction
- Construction and composition
- Substrings: creation and lookup
- Conversion between strings and numbers
- **Practice**
- Summary
- Home exercise

Practice: is string a palindrome?

Write a function that returns `true` when the input string is a palindrome and `false` otherwise

```
bool is_palindrome(const std::string &word);

TEST_CASE("palindromes") {
    CHECK(is_palindrome("level"));
    CHECK(is_palindrome("reviver"));

    CHECK_FALSE(is_palindrome("persimmon"));

    CHECK(is_palindrome(""));
    CHECK(is_palindrome("I"));
}
```

Practice: a possible solution

```cpp
bool is_palindrome(const std::string &word) {
    const std::string reversed_word{word.crbegin(), word.crend()};
    return word == reversed_word;
}
```

std::string: Session plan

- Recap and updates
- Introduction
- Construction and composition
- Substrings: creation and lookup
- Conversion between strings and numbers
- Practice
- **Summary**
- Home exercise

std::string: Summary

std::string is a library type for efficient string operations

- It’s quite similar to a vector of characters
- It doesn’t have any knowledge of Unicode :-(
- It has additional methods for composition, slicing and substring lookup
- The standard library provides conversion functions between strings and numbers

std::string: Session plan

- Recap and updates
- Introduction
- Construction and composition
- Substrings: creation and lookup
- Conversion between strings and numbers
- Practice
- Summary
- **Home exercise**

Home exercise: count substring occurrences

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);
}
```

Thank You!

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