C++ libraries for daily use

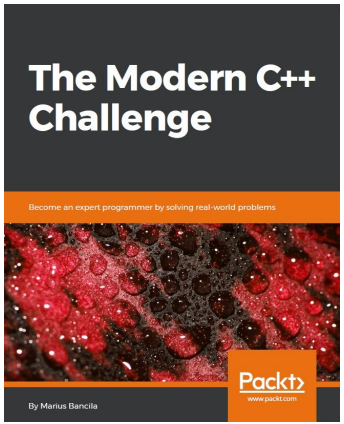
Marius Bancila

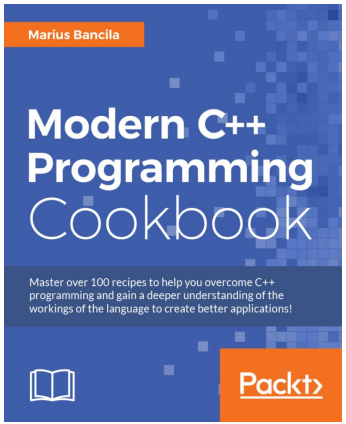
https://mariusbancila.ro

https://github.com/mariusbancila

mariusbancila

marius.bancila





fmt

https://github.com/fmtlib/fmt

fmtlib / **fmt**

Watch 284 | Star 7.7k | Fork 941

Code | Issues 14 | Pull requests 2 | Actions | Projects 0 | Security | Insights

A modern formatting library https://fmt.dev

c-plus-plus formatting printf

4,022 commits | 3 branches | 0 packages | 25 releases | 183 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

IkarusDeveloper improved use of find (#1560) ... Latest commit e00997b 2 days ago

.github	Create PR template to help agree to the licensing terms (#1083)	11 months ago
doc	Add locale example	last month
include/fmt	improved use of find (#1560)	2 days ago
src	Remove misleading FMT_USE_WINDOWS_H	9 days ago

https://github.com/fmtlib/fmt

fmtlib / **fmt**

Watch 284 | Star 7.7k | Fork 941

Code | Issues 14 | Pull requests 2 | Actions | Projects 0 | Security | Insights

A modern formatting library https://fmt.dev

c-plus-plus formatting printf

4,022 commits | 3 branches | 0 packages | 25 releases | 183 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

IkarusDeveloper improved use of find (#1560) ... Latest commit e00997b 2 days ago

.github	Create PR template to help agree to the licensing terms (#1083)	11 months ago
doc	Add locale example	last month
include/fmt	improved use of find (#1560)	2 days ago
src	Remove misleading FMT_USE_WINDOWS_H	9 days ago

Intro

- Modern formatting library
- Open-source https://github.com/fmtlib/fmt
- Safe alternative to printf and streams
  - Fully type safe, errors in format strings reported at compile time
- Support for used-defined types
- High-performance (faster than printf and streams)
- Implements C++20 std::format
- Support for wide strings
- Lib or headers only

Victor (negative 🦄) Zverovich
@vzverovich

{fmt} is now ~35% faster than printf on tinyformat speed test (previously it was "only" ~10% faster) and almost 4x faster than cout. 🚀

4:51 PM · Nov 22, 2019 · Twitter for Android

23 Retweets **163** Likes

Getting started

Includes

```
#define FMT_HEADER_ONLY 1
#include "fmt/format.h"
#include "fmt/core.h"
#include "fmt/ostream.h"
#include "fmt/printf.h"
#include "fmt/chrono.h"
```

Examples

C++

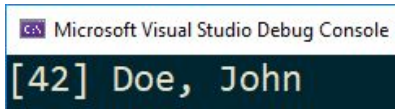
```cpp
auto text = fmt::format("[{}] {}, {}", 42, "Doe", "John");
std::cout << text << '\n';
```

C#

```csharp
var text = string.Format("[{0}] {2}, {1}", 42, "John", "Doe");
Console.WriteLine(text);
```

Python

```python
text = '[{}] {}, {}'.format(42, "Doe", "John")
print(text)
```



Examples: positional arguments

C++

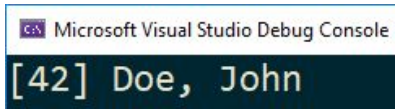
```cpp
auto text = fmt::format("[{0}] {2}, {1}", 42, "John", "Doe");
std::cout << text << '\n';
```

C#

```cs
var text = string.Format("[{0}] {2}, {1}", 42, "John", "Doe");
Console.WriteLine(text);
```

Python

```python
text = '[{0}] {2}, {1}'.format(42, "John", "Doe")
print(text)
```



Examples: named arguments

C++

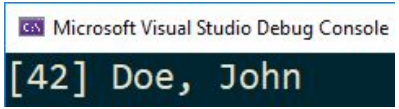
```cpp
auto text = fmt::format("[{id}] {lastName}, {firstName}",
                        fmt::arg("id", 42), fmt::arg("firstName", "John"),
                        fmt::arg("lastName", "Doe"));
std::cout << text << '\n';
```

C#

```csharp
id = 42;
string firstName="John", lastName="Doe";
var text = $"[{id}] {firstName}, {lastName}";
Console.WriteLine(text);
```

Python

```python
text = "[{id}] {lastName}, {firstName}".format(id=42, firstName="John",
                                               lastName="Doe")
print(text)
```



Examples: named arguments + user-defined literals

C++

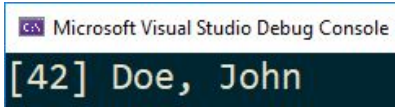
```cpp
using namespace fmt::literals;
auto text = fmt::format("[{id}] {lastName}, {firstName}",
                        "id"_a=42, "firstName"_a="John", "lastName"_a="Doe");
std::cout << text << '\n';
```

C#

```csharp
id = 42;
string firstName="John", lastName="Doe";
var text = $"[{id}] {firstName}, {lastName}";
Console.WriteLine(text);
```

Python

```python
text = "[{id}] {lastName}, {firstName}".format(id=42, firstName="John",
                                               lastName="Doe")
print(text)
```

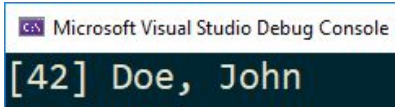


Examples: with user-defined literals

C++

```
using namespace fmt::literals;
auto text = "[{}] {}, {}"_format(42, "Doe", "John");
std::cout << text << '\n';
```

```
using namespace fmt::literals;
auto text = "[{0}] {2}, {1}"_format(42, "John", "Doe");
std::cout << text << '\n';
```



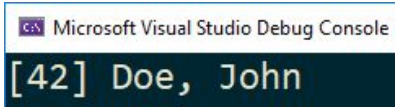
Formatting APIs

`fmt::format()`	`std::string`
`fmt::format_to()`	`fmt::memory_buffer` / `fmt::wmemory_buffer`
`fmt::print()`	stdout / file (`FILE*`)

`fmt::printf()`	`stdout`
`fmt::fprintf()`	File (`FILE*`) / stream (`std::basic_ostream`)
`fmt::sprintf()`	`std::string`

Examples: format to memory buffer

```
fmt::memory_buffer buf;
fmt::format_to(buf, "[{0}] {2}, {1}", 42, "John", "Doe");
std::cout << buf.data() << '\n';
```

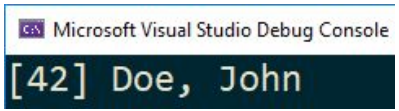


Examples: print to standard output

```
fmt::print("[{}] {}, {}\n", 42, "Doe", "John");
```

```
fmt::print("[{0}] {2}, {1}\n", 42, "John", "Doe");
```

```
using namespace fmt::literals;
fmt::print("[{id}] {lastName}, {firstName}\n",
           "id"_a=42, "firstName"_a="John", "lastName"_a="Doe");
```

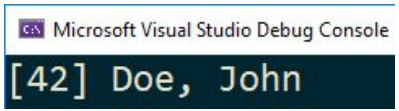


Examples: print to standard error

```
fmt::print(stderr, "[{}] {}, {}\n", 42, "Doe", "John");
```

```
fmt::print(stderr, "[{0}] {2}, {1}\n", 42, "John", "Doe");
```

```
using namespace fmt::literals;
fmt::print(stderr, "[{id}] {lastName}, {firstName}\n",
           "id"_a=42, "firstName"_a="John", "lastName"_a="Doe");
```

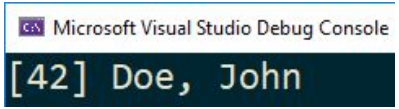


Examples: print to standard error

```
fmt::print(std::cerr, "[{}] {}, {}\n", 42, "Doe", "John");
```

```
fmt::print(std::cerr, "[{0}] {2}, {1}\n", 42, "John", "Doe");
```

```
using namespace fmt::literals;
fmt::print(std::cerr, "[{id}] {lastName}, {firstName}\n",
           "id"_a=42, "firstName"_a="John", "lastName"_a="Doe");
```



Examples: printf-like syntax

Standard output

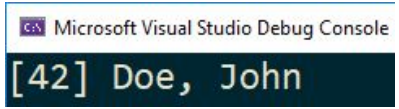
```
fmt::printf("[%d] %s, %s\n", 42, "Doe", "John");
```

File

```
fmt::fprintf(stderr, "[%d] %s, %s\n", 42, "Doe", "John");
```

String

```
auto text = fmt::sprintf("[%d] %s, %s", 42, "Doe", "John");
std::cout << text << '\n';
```



Examples: date & time support

```
std::time_t t = std::time(nullptr);
fmt::print("Today is {:%Y-%m-%d}\n", *std::localtime(&t));
```

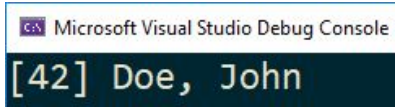
User types formatting

Type

```
struct employee
{
   int         id;
   std::string firstName;
   std::string lastName;
};
```

Formatting

```
employee e { 42, "John", "Doe" };
fmt::print("{}\n", e);
```



User types formatting

Formatter

```
template<>
struct fmt::formatter<employee>
{
   template <typename ParseContext>
   constexpr auto parse(ParseContext& ctx) { return ctx.begin(); }

   template <typename FormatContext>
   auto format(employee const & e, FormatContext& ctx)
   {
      return fmt::format_to(ctx.out(), "[{}] {}, {}", e.id, e.lastName, e.firstName);
   }
};
```

User types formatting

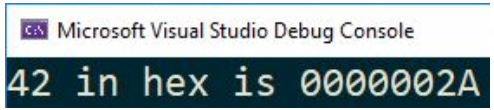
```
struct employee
{
   int         id;
   std::string firstName;
   std::string lastName;
};
```

```
employee e { 42, "John", "Doe" };
fmt::print("{}\n", e);
```

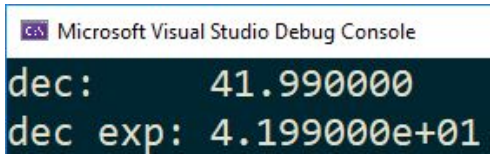
```
template<>
struct fmt::formatter<employee>
{
   template <typename ParseContext>
   constexpr auto parse(ParseContext& ctx) { return ctx.begin(); }

   template <typename FormatContext>
   auto format(employee const & e, FormatContext& ctx)
   {
      return fmt::format_to(ctx.out(), "[{}] {}, {}", e.id, e.lastName, e.firstName);
   }
};
```

Examples: format specifiers

```
fmt::print("42 in hex is {:08X}\n", 42);
```




```
fmt::print("dec:     {0:f}\ndec exp: {0:e}\n", 41.99);
```



Safety: runtime

argument index out of range

```
try
{
   fmt::print("[{}] {}, {}\n", 42, "Doe");
}
catch (fmt::format_error const& e)
{
   fmt::print("{}\n", e.what());
}
```

invalid type specifier

```
try
{
   fmt::print("The answer is {:d}", "forty-two");
}
catch (fmt::format_error const& e)
{
   fmt::print("{}\n", e.what());
}
```

Safety: compile time

mixing character types is disallowed

```
fmt::print("{}", L'\x42e');
```

expression did not evaluate to a constant

```
fmt::print(FMT_STRING("The answer is {:d}"), "forty-two");
```

Requirements

- C++11 features
  - variadic templates
  - type traits
  - rvalue references
  - decltype
  - trailing return types
  - deleted functions
  - alias templates
- Availability
  - GCC 4.8
  - Clang 3.0
  - MSVC 19.0 (2015).

Standard formatting library C++20

C++20

```
auto text = std::format("[{}] {}, {}", 42, "Doe", "John");
std::cout << text << '\n';
```

```
auto text = std::format("{0} hex is {0:08X}", 42);

auto now = std::chrono::system_clock::now();
auto time = std::chrono::system_clock::to_time_t(now);
auto text = std::format("Today is {:%Y-%m-%d}",
                        *std::localtime(&time));

std::vector<char> buf;
std::format_to(std::back_inserter(buf), "{} is {}", "John", 42);
```

loguru

https://github.com/emilk/loguru

emilk / loguru

Watch 47 | Star 834 | Fork 127

Code | Issues 23 | Pull requests 4 | Actions | Projects 0 | Wiki | Security | Insights

A lightweight C++ logging library

265 commits | 2 branches | 0 packages | 7 releases | 30 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

oriolarcas and emilk Check isatty() to decide if colors are supported (#127) ... Latest commit 2e1424d on Jan 13

docs	feature: optional header log with columns name	3 months ago
glog_bench	Improve benchmarks with warmup	4 years ago
glog_example	Add some example output of GLOG and Loguru to compare the two.	4 years ago
loguru_bench	Only compile loguru_bench.cpp for loguru_bench	14 months ago

https://github.com/emilk/loguru

emilk / loguru

Watch 47 | Star 834 | Fork 127

Code | Issues 23 | Pull requests 4 | Actions | Projects 0 | Wiki | Security | Insights

A lightweight C++ logging library

265 commits | 2 branches | 0 packages | 7 releases | 30 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

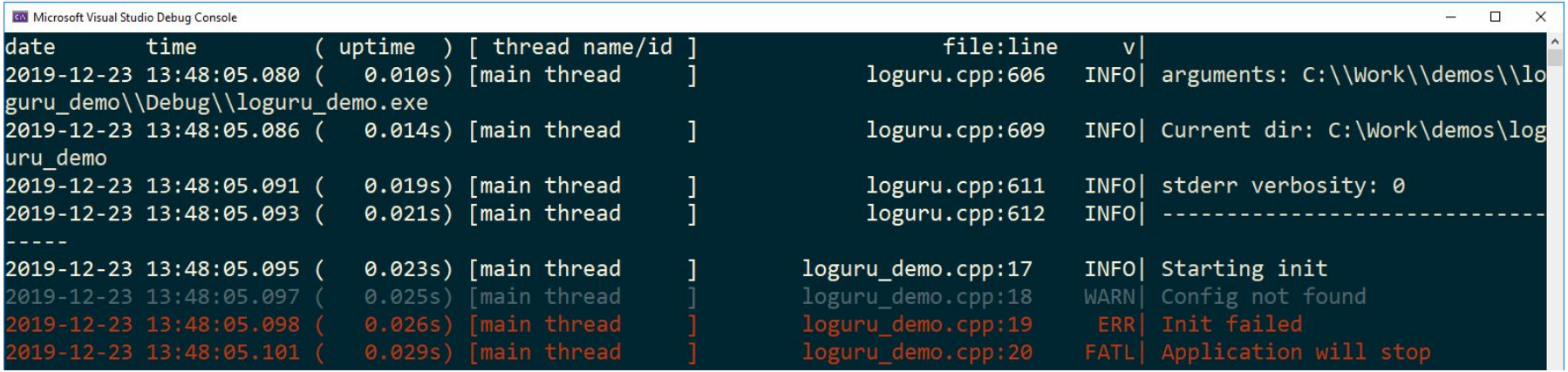
oriolarcas and emilk Check isatty() to decide if colors are supported (#127) ... Latest commit 2e1424d on Jan 13

docs	feature: optional header log with columns name	3 months ago
glog_bench	Improve benchmarks with warmup	4 years ago
glog_example	Add some example output of GLOG and Loguru to compare the two.	4 years ago
loguru_bench	Only compile loguru_bench.cpp for loguru_bench	14 months ago

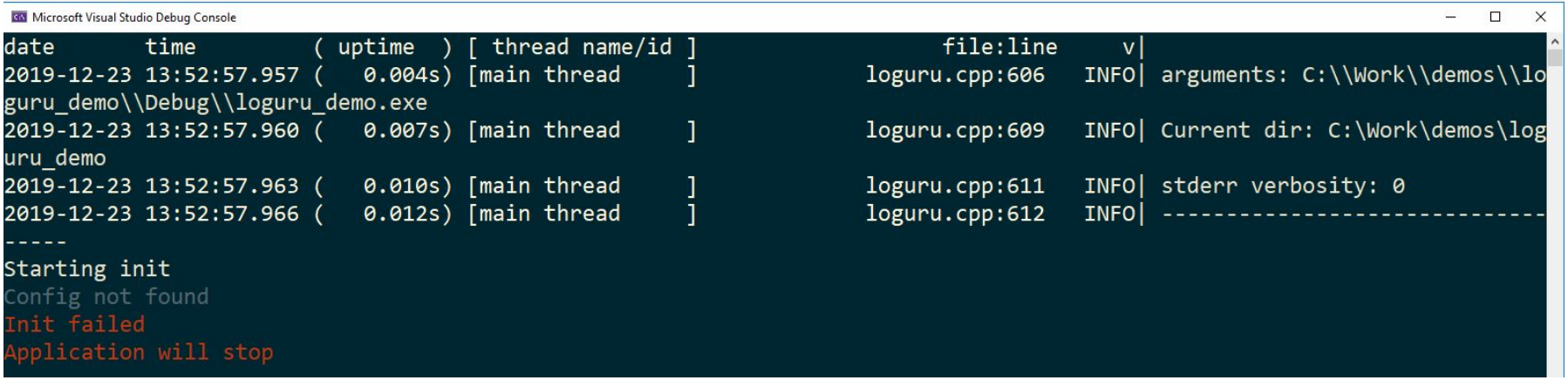
Intro

- Open-source https://github.com/emilk/loguru
- Small, simple, flexible
- Features
  - Verbosity levels
  - Assertions
  - Aborts
  - Stack traces printed on abort
  - Formatting styles: printf, streams, fmtlib
  - Prefix for log lines (time, uptime, thread, file, line, log level)
  - Scopes
  - grep:able content

```cpp
#include "loguru.hpp"

int main(int argc, char* argv[])
{
    loguru::init(argc, argv);

    LOG_F(INFO,    "Starting init");
    LOG_F(WARNING, "Config not found");
    LOG_F(ERROR,   "Init failed");
    LOG_F(FATAL,   "Application will stop");
}
```



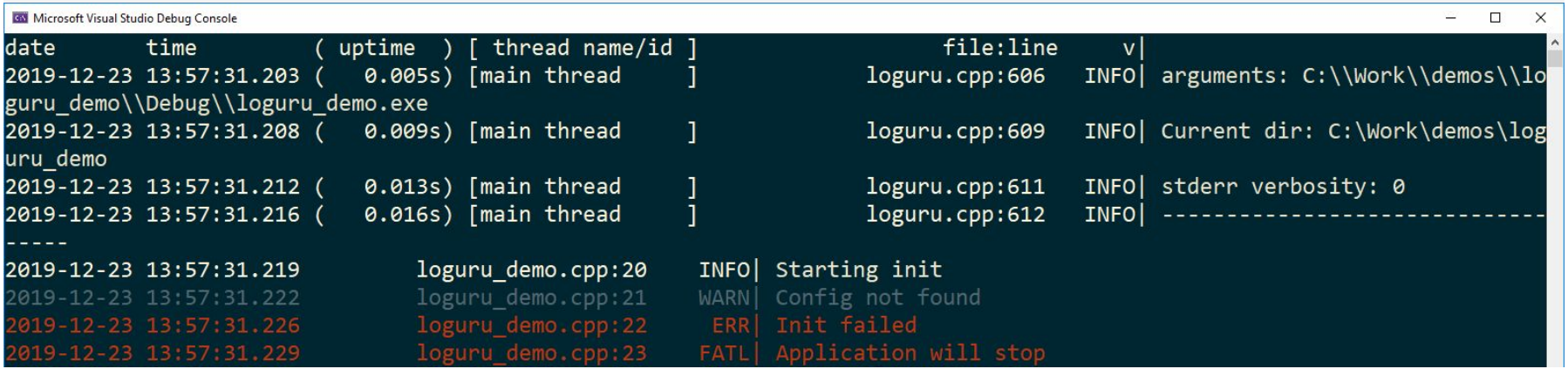
```cpp
#include "loguru.hpp"

int main(int argc, char* argv[])
{
    loguru::init(argc, argv);
    loguru::g_preamble = false;

    LOG_F(INFO,    "Starting init");
    LOG_F(WARNING, "Config not found");
    LOG_F(ERROR,   "Init failed");
    LOG_F(FATAL,   "Application will stop");
}
```



Turn off preamble parts

```
// Turn off individual parts of the preamble
LOGURU_EXPORT extern bool      g_preamble_date;    // The date field
LOGURU_EXPORT extern bool      g_preamble_time;    // The time of the current day
LOGURU_EXPORT extern bool      g_preamble_uptime;  // The time since init call
LOGURU_EXPORT extern bool      g_preamble_thread;  // The logging thread
LOGURU_EXPORT extern bool      g_preamble_file;    // The file from which the log originates from
LOGURU_EXPORT extern bool      g_preamble_verbose; // The verbosity field
LOGURU_EXPORT extern bool      g_preamble_pipe;    // The pipe symbol right before the message
```

```cpp
#include "loguru.hpp"

int main(int argc, char* argv[])
{
    loguru::init(argc, argv);
    loguru::g_preamble_thread = false;
    loguru::g_preamble_uptime = false;

    LOG_F(INFO,    "Starting init");
    LOG_F(WARNING, "Config not found");
    LOG_F(ERROR,   "Init failed");
    LOG_F(FATAL,   "Application will stop");
}
```



Other settings

```
LOGURU_EXPORT extern Verbosity g_internal_verbosity;// Specify the verbosity used by loguru to log
                                                    // its info messages including the header
                                                    // logged when logged::init() is called or on
                                                    // exit. Default is 0 (INFO).

LOGURU_EXPORT extern Verbosity g_stderr_verbosity;  // Everything with a verbosity equal or greater
                                                    // than g_stderr_verbosity will be written to
                                                    // stderr.

LOGURU_EXPORT extern bool      g_colorlogtostderr;  // True by default.

LOGURU_EXPORT extern unsigned  g_flush_interval_ms; // 0 (unbuffered) by default.

LOGURU_EXPORT extern bool      g_preamble;          // Prefix each log line with date, time etc?
                                                    // True by default.
```

```cpp
#include "loguru.hpp"

int main(int argc, char* argv[])
{
    loguru::init(argc, argv);
    loguru::g_preamble_thread = false;
    loguru::g_preamble_uptime = false;
    loguru::g_stderr_verbosity = loguru::Verbosity_ERROR;
    loguru::g_colorlogtostderr = false;

    LOG_F(INFO,    "Starting init");
    LOG_F(WARNING, "Config not found");
    LOG_F(ERROR,   "Init failed");
    LOG_F(FATAL,   "Application will stop");
}
```

Microsoft Visual Studio Debug Console

```
date       time         ( uptime  ) [ thread name/id ]                  file:line     v|
2019-12-23 14:02:29.414 (   0.005s) [main thread     ]            loguru.cpp:606   INFO| arguments: C:\\Work\\demos\\lo
guru_demo\\Debug\\loguru_demo.exe
2019-12-23 14:02:29.418 (   0.007s) [main thread     ]            loguru.cpp:609   INFO| Current dir: C:\Work\demos\log
uru_demo
2019-12-23 14:02:29.421 (   0.011s) [main thread     ]            loguru.cpp:611   INFO| stderr verbosity: 0
2019-12-23 14:02:29.423 (   0.013s) [main thread     ]            loguru.cpp:612   INFO| ----------------------------
-----
2019-12-23 14:02:29.428         loguru_demo.cpp:24     ERR| Init failed
2019-12-23 14:02:29.430         loguru_demo.cpp:25    FATL| Application will stop
```

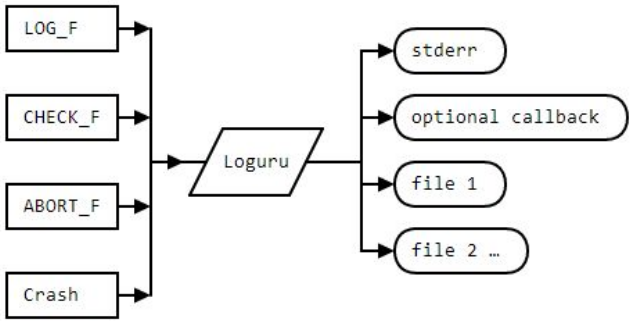
Verbosity levels

- `FATAL` (-3)
- `ERROR` (-2)
- `WARNING` (-1)
- `INFO` (0)
- 1-9 (custom)

- Only messages with verbosity lower or equal than the verbosity of the output are processed

How it works

- `stderr` is the default output
- add files with `loguru::add_file`
- each output has a verbosity level attached
  - Everything equal or lower than the verbosity level is written
- Control `stderr` verbosity level
  - From code with `loguru::g_stderr_verbosity`
  - From command line with `-v`



```cpp
#include "loguru.hpp"

int main(int argc, char* argv[])
{
    loguru::g_preamble_thread = false;
    loguru::g_preamble_uptime = false;
    loguru::init(argc, argv);
    loguru::add_file("important.log", loguru::Append, loguru::Verbosity_WARNING);
    loguru::add_file("everything.log", loguru::Append, loguru::Verbosity_MAX);

    LOG_F(INFO,    "Starting init");
    LOG_F(WARNING, "Config not found");
    LOG_F(ERROR,   "Init failed");
    LOG_F(FATAL,   "Application will stop");
}
```

Microsoft Visual Studio Debug Console

```
date       time                          file:line     v|
2019-12-23 14:24:06.353              loguru.cpp:606   INFO| arguments: C:\\Work\\demos\\loguru_demo\\Debug\\loguru_demo.
exe
2019-12-23 14:24:06.356              loguru.cpp:609   INFO| Current dir: C:\Work\demos\loguru_demo
2019-12-23 14:24:06.358              loguru.cpp:611   INFO| stderr verbosity: 0
2019-12-23 14:24:06.360              loguru.cpp:612   INFO| ----------------------------------
2019-12-23 14:24:06.365              loguru.cpp:785   INFO| Logging to 'important.log', mode: 'a', verbosity: -1
2019-12-23 14:24:06.368              loguru.cpp:785   INFO| Logging to 'everything.log', mode: 'a', verbosity: 9
2019-12-23 14:24:06.371        loguru_demo.cpp:22    INFO| Starting init
2019-12-23 14:24:06.374        loguru_demo.cpp:23    WARN| Config not found
2019-12-23 14:24:06.376        loguru_demo.cpp:24     ERR| Init failed
2019-12-23 14:24:06.377        loguru_demo.cpp:25    FATL| Application will stop
```

important.log

```
arguments: C:\\Work\\demos\\loguru_demo\\Debug\\loguru_demo.exe
Current dir: C:\Work\demos\loguru_demo
File verbosity level: -1
date       time                           file:line     v|
2019-12-23 14:24:06.374         loguru_demo.cpp:23    WARN| Config not found
2019-12-23 14:24:06.376         loguru_demo.cpp:24     ERR| Init failed
2019-12-23 14:24:06.377         loguru_demo.cpp:25    FATL| Application will stop
```

everything.log

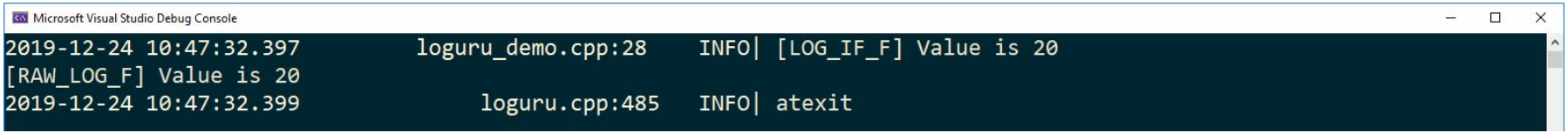
```
arguments: C:\\Work\\demos\\loguru_demo\\Debug\\loguru_demo.exe
Current dir: C:\Work\demos\loguru_demo
File verbosity level: 9
date       time                           file:line     v|
2019-12-23 14:24:06.368              loguru.cpp:785   INFO| Logging to 'everything.log', mode: 'a', verbosity: 9
2019-12-23 14:24:06.371         loguru_demo.cpp:22    INFO| Starting init
2019-12-23 14:24:06.374         loguru_demo.cpp:23    WARN| Config not found
2019-12-23 14:24:06.376         loguru_demo.cpp:24     ERR| Init failed
2019-12-23 14:24:06.377         loguru_demo.cpp:25    FATL| Application will stop
```

Logging functions

All functions	**Debug only** (LOGURU_DEBUG_CHECKS)	**Description**
LOG_F(verbosity_name, fmt, ...)	**DLOG_F**(verbosity_name, fmt, ...)	Standard logging function
VLOG_F(verbosity, fmt, ...)	**DVLOG_F**(verbosity, fmt, ...)	Dynamic verbosity (from a function call)
LOG_IF_F(verbosity_name, cond, fmt, ...)	**DLOG_IF_F**(verbosity_name, cond, fmt, ...)	Conditional logging
VLOG_IF_F(verbosity, cond, fmt, ...)	**DVLOG_IF_F**(verbosity, cond, fmt, ...)	Conditional logging with dynamic verbosity
RAW_LOG_F(verbosity_name, fmt, ...)	**DRAW_LOG_F**(verbosity_name, fmt, ...)	Logging without preamble
RAW_VLOG_F(verbosity, fmt, ...)	**DRAW_VLOG_F**(verbosity, fmt, ...)	Logging without preamble with dynamic verbosity

Example

```
int value = 20;
LOG_IF_F(INFO, value < 42, "[LOG_IF_F] Value is %d", value);

RAW_LOG_F(INFO, "[RAW_LOG_F] Value is %d", value);
```

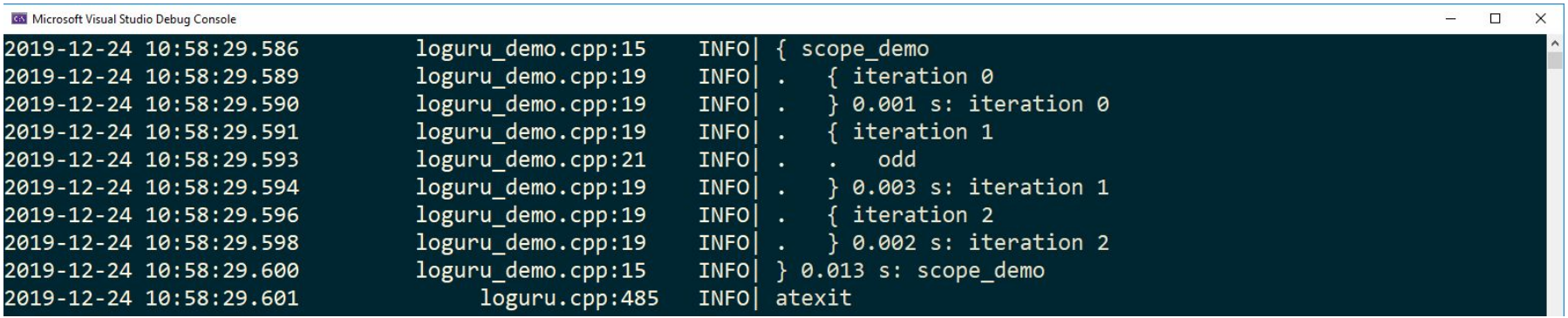


Logging scope functions

All functions	Debug only	Description
`LOG_SCOPE_F(verbosity_name, fmt, ...)`		Logs iteration, execution type
`VLOG_SCOPE_F(verbosity, fmt, ...)`		Logs iteration with dynamic verbosity
`LOG_SCOPE_FUNCTION(verbosity_name)`		Logs the name of the current function

Example

```
void scope_demo(int const count)
{
   LOG_SCOPE_FUNCTION(INFO);
   for (int i = 0; i < count; ++i)
   {
      LOG_SCOPE_F(INFO, "iteration %d", i);
      LOG_IF_F(INFO, i % 2 == 1, "odd");
   }
}

scope_demo(3);
```



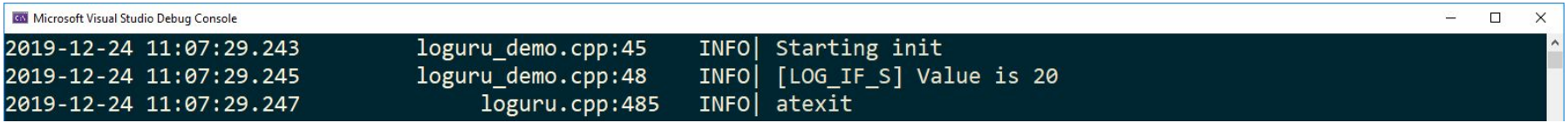
Logging with streams

All functions	Debug only	Description
`LOG_S(verbosity_name) << ...`	`DLOG_S(verbosity_name) << ...`	Standard logging function
`VLOG_S(verbosity) << ...`	`DVLOG_S(verbosity) << ...`	Dynamic verbosity (from a function call)
`LOG_IF_S(verbosity_name, cond) << ...`	`DLOG_IF_S(verbosity_name, cond) << ...`	Conditional logging
`VLOG_IF_S(verbosity, cond) << ...`	`DVLOG_IF_S(verbosity, cond) << ...`	Conditional logging with dynamic verbosity

- `#define LOGURU_WITH_STREAMS 1`, or
- Pass `-DLOGURU_WITH_STREAMS=1` to the compiler

Example

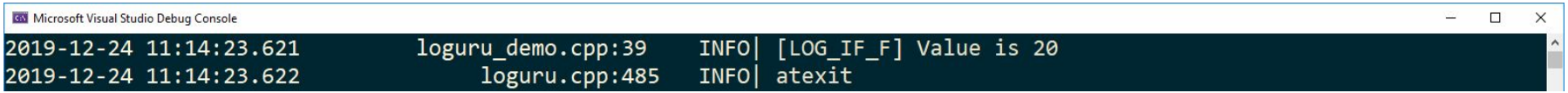
```
LOG_S(INFO) << "Starting init";

int value = 20;
LOG_IF_S(INFO, value < 42) << "[LOG_IF_S] Value is " << value;
```



Logging with fmtlib

- `#define LOGURU_USE_FMTLIB 1`, or
- Pass `-DLOGURU_USE_FMTLIB=1` to the compiler

```
#define LOGURU_USE_FMTLIB 1
#define FMT_HEADER_ONLY 1
#include "fmt\format.h"

int value = 20;
LOG_IF_F(INFO, value < 42, "[LOG_IF_F] Value is {}", value);
```

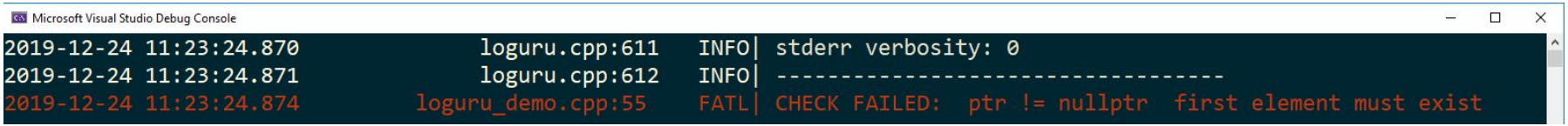


Checking functions

All functions	Debug only	Check that
`CHECK_F(condition, ...)`	`DCHECK_F(condition, ...)`	`condition` evaluates to true
`CHECK_NOTNULL_F(ptr, ...)`	`DCHECK_NOTNULL_F(ptr, ...)`	`ptr` is not null
`CHECK_EQ_F(a, b, ...)`	`DCHECK_EQ_F(a, b, ...)`	`a == b`
`CHECK_NE_F(a, b, ...)`	`DCHECK_NE_F(a, b, ...)`	`a != b`
`CHECK_LT_F(a, b, ...)`	`DCHECK_LT_F(a, b, ...)`	`a < b`
`CHECK_LE_F(a, b, ...)`	`DCHECK_LE_F(a, b, ...)`	`a <= b`
`CHECK_GT_F(a, b, ...)`	`DCHECK_GT_F(a, b, ...)`	`a > b`
`CHECK_GE_F(a, b, ...)`	`DCHECK_GE_F(a, b, ...)`	`a >= b`

Example

```
std::vector<int> data{ 1,2,3 };
CHECK_F(!data.empty(), "must not be empty");

int* ptr = nullptr;
CHECK_NOTNULL_F(ptr, "first element must exist");
```



And more...

- Functions that control the logging
- Logging callbacks
- Fatal handler
- Runtime options
- Compile-time options

Lyra

https://github.com/bfgroup/Lyra

bfgroup / Lyra

forked from catchorg/Clara

Watch 7 | Unstar 101 | Fork 81

Code | Issues 1 | Pull requests 0 | Actions | Projects 1 | Security | Insights

A simple to use, composable, command line parser for C++ 11 and beyond https://bfgroup.github.io/Lyra/

cpp cpp11 cpp14 cpp17 cpp20 cli argument-parser argument-parsing cmdline-parser options-parsing option-parser header-only

296 commits | 2 branches | 0 packages | 4 releases | 16 contributors | BSL-1.0

Branch: develop | New pull request | Create new file | Upload files | Find file | Clone or download

This branch is 158 commits ahead, 3 commits behind catchorg:master. | Pull request | Compare

jayeshbadwaik Namespace cli_parser for correct usage. (#20) — Latest commit 4235801 1 hour ago

.vscode	Some cleanup of parse results.	3 months ago
data	Start 1.4	last month
docs	Start 1.4	last month
examples	Add tests for the rest of the doc examples.	last month
include/lyra	Start 1.4	last month

https://github.com/bfgroup/Lyra

bfgroup / **Lyra**

forked from catchorg/Clara

Watch 7 | Unstar 101 | Fork 81

Code | Issues 1 | Pull requests 0 | Actions | Projects 1 | Security | Insights

A simple to use, composable, command line parser for C++ 11 and beyond https://bfgroup.github.io/Lyra/

cpp cpp11 cpp14 cpp17 cpp20 cli argument-parser argument-parsing cmdline-parser options-parsing option-parser header-only

296 commits | 2 branches | 0 packages | 4 releases | 16 contributors | BSL-1.0

Branch: develop | New pull request | Create new file | Upload files | Find file | Clone or download

This branch is 158 commits ahead, 3 commits behind catchorg:master. | Pull request | Compare

jayeshbadwaik Namespace cli_parser for correct usage. (#20) — Latest commit 4235801 1 hour ago

.vscode	Some cleanup of parse results.	3 months ago
data	Start 1.4	last month
docs	Start 1.4	last month
examples	Add tests for the rest of the doc examples.	last month
include/lyra	Start 1.4	last month

Intro

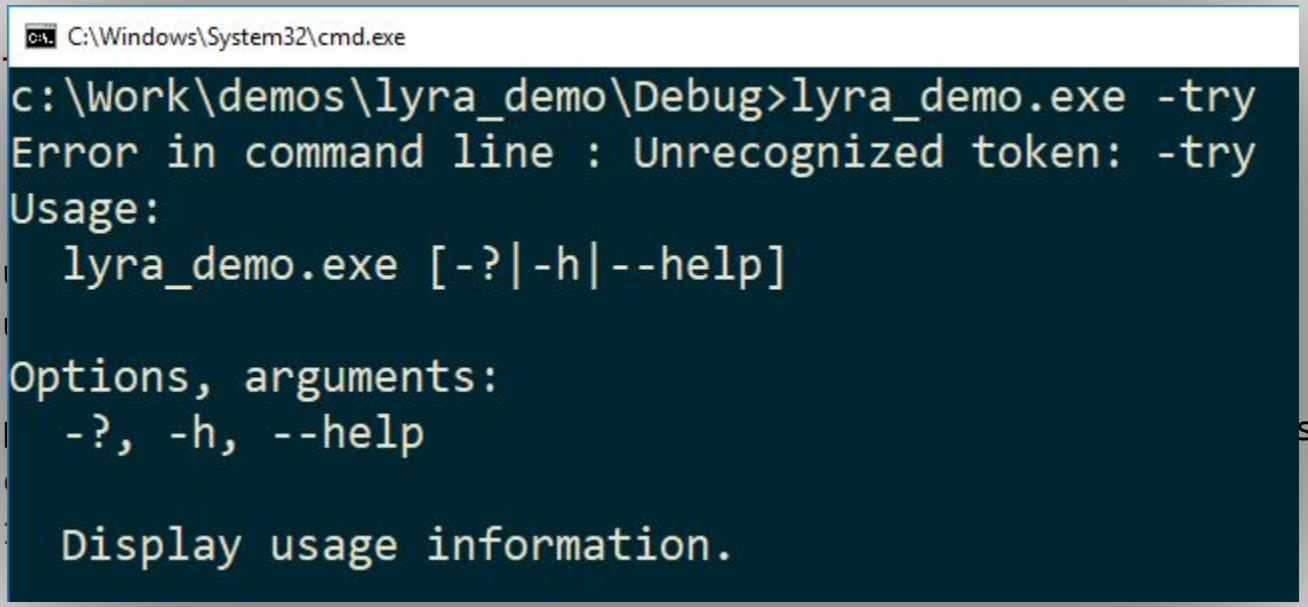
- Command line argument parser for C++11 (and beyond)
- Multi-sources or header only
- Cross-platform, open-source
- Features
  - Optional arguments
  - Mandatory arguments
  - Binding arguments to variables
  - Usage / help
  - Support for user-defined types
  - Custom parsing

Getting started

```
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
   auto cli = lyra::cli_parser() | ... ;

   auto result = cli.parse({ argc, argv });
   if (!result)
   {
      fmt::print(std::cerr, "Error in command line : {}\n", result.errorMessage());
      std::cerr << cli;
      exit(1);
   }
}
```

Getting started

```
#include "lyra/lyra.hpp"

int main(int
{
    auto cli

    auto resu
    if (!resu
    {
        fmt::p                                    ssage());
        std::c
        exit(1
    }
}
```

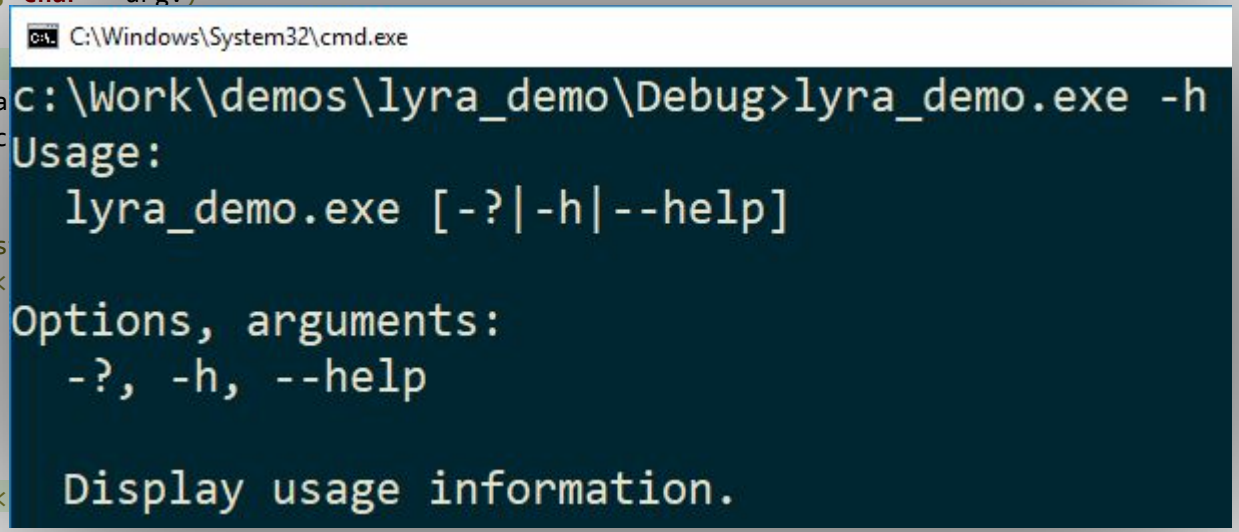


Show usage

```cpp
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
   auto show_help = false;
   auto cli = lyra::cli_parser() | lyra::help(show_help);
   auto result = cli.parse({ argc, argv });
   if (!result)
   {
      fmt::print(std::cerr, "Error in command line : {}\n", result.errorMessage());
      std::cerr << cli;
      exit(1);
   }

   if (show_help)
   {
      std::cout << cli;
      exit(0);
   }
}
```

Show usage

```
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
    auto show_help
    auto cli = lyra
    auto result = c
    if (!result)
    {
        fmt::print(s
        std::cerr <<
        exit(1);
    }

    if (show_help)
    {
        std::cout <<
        exit(0);
    }
}
```

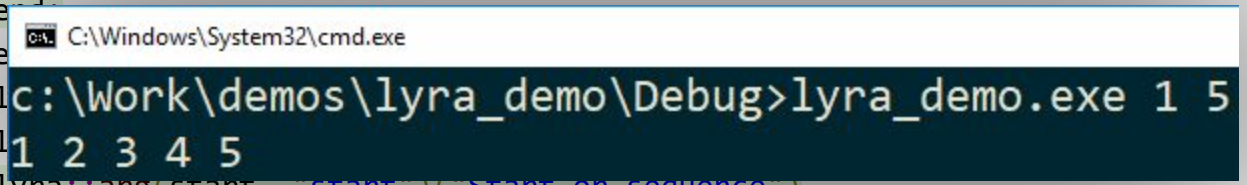


Mandatory (positional) arguments

```cpp
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
   int start, end;
   auto show_help = false;
   auto cli = lyra::cli_parser()
            | lyra::help(show_help)
            | lyra::arg(start, "start")("Start or sequence")
            | lyra::arg(end, "end")("End of sequence");

   auto result = cli.parse({ argc, argv });
   /* ... */

   for (int i = start; i <= end; ++i)
      fmt::print("{} ", i);
}
```

Mandatory (positional) arguments

```cpp
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
    int start, end;
    auto show_he
    auto cli = l
             | l
             | lyra::arg(start, "start")("Start of sequence")
             | lyra::arg(end, "end")("End of sequence");

    auto result = cli.parse({ argc, argv });
    /* ... */

    for (int i = start; i <= end; ++i)
        fmt::print("{} ", i);
}
```

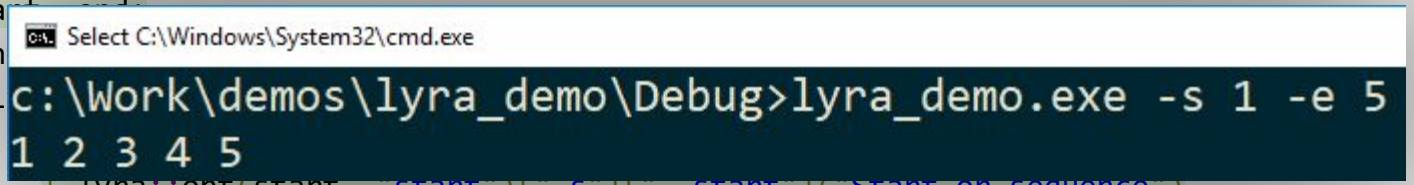


Optional arguments

```cpp
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
   int start, end;
   auto show_help = false;
   auto cli = lyra::cli_parser()
            | lyra::help(show_help)
            | lyra::opt(start, "start")["-s"]["--start"]("Start or sequence")
            | lyra::opt(end, "end")["-e"]["--end"]("End of sequence");

   auto result = cli.parse({ argc, argv });
   /* ... */

   for (int i = start; i <= end; ++i)
      fmt::print("{} ", i);
}
```

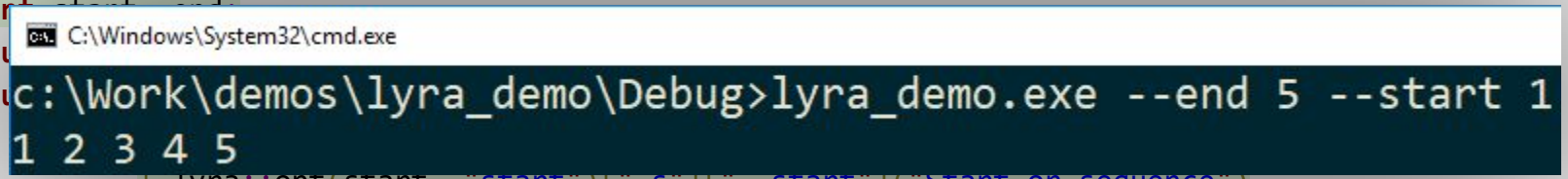
Optional arguments

```cpp
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
    int start, end;
    auto sh
    auto cl
            | lyra::opt(start, "start")["-s"]["--start"]("Start of sequence")
            | lyra::opt(end, "end")["-e"]["--end"]("End of sequence");

    auto result = cli.parse({ argc, argv });
    /* ... */

    for (int i = start; i <= end; ++i)
        fmt::print("{} ", i);
}
```



Optional arguments

```
#include "lyra/lyra.hpp"

int main(int argc, char** argv)
{
    int start, end;
    auto cli = lyra::opt(start, "start")["-s"]["--start"]("Start of sequence")
             | lyra::opt(end, "end")["-e"]["--end"]("End of sequence");

    auto result = cli.parse({ argc, argv });
    /* ... */

    for (int i = start; i <= end; ++i)
        fmt::print("{} ", i);
}
```



Optional arguments

```
#include "lyra/lyra
int main(int argc,
{
    int start, end;
    auto show_help =
    auto cli = lyra:
               | lyra:
               | lyra:
               | lyra:
    auto result = cl
    /* ... */
    for (int i = sta
        fmt::print("{
}
```

```
C:\Windows\System32\cmd.exe
c:\Work\demos\lyra_demo\Debug>lyra_demo.exe -h
Usage:
  lyra_demo.exe [-?|-h|--help] [-s|--start] [-e|--end]

Options, arguments:
  -?, -h, --help

  Display usage information.

  -s, --start <start>

  Start or sequence

  -e, --end <end>

  End of sequence
```

Flag arguments

```
int main(int argc, char** argv)
{
   int start = 0, end = 0;
   auto show_help = false;
   auto add_nl = false;
   auto cli = lyra::cli_parser()
            | lyra::help(show_help)
            | lyra::opt(start, "start")["-s"]["--start"]("Start or sequence")
            | lyra::opt(end, "end")["-e"]["--end"]("End of sequence")
            | lyra::opt(add_nl)["-l"]["--newline"]("Add new line");

   auto result = cli.parse({ argc, argv });
   /* ... */

   for (int i = start; i <= end; ++i)
      fmt::print("{} ", i);
   if (add_nl) fmt::print("\n");
}
```

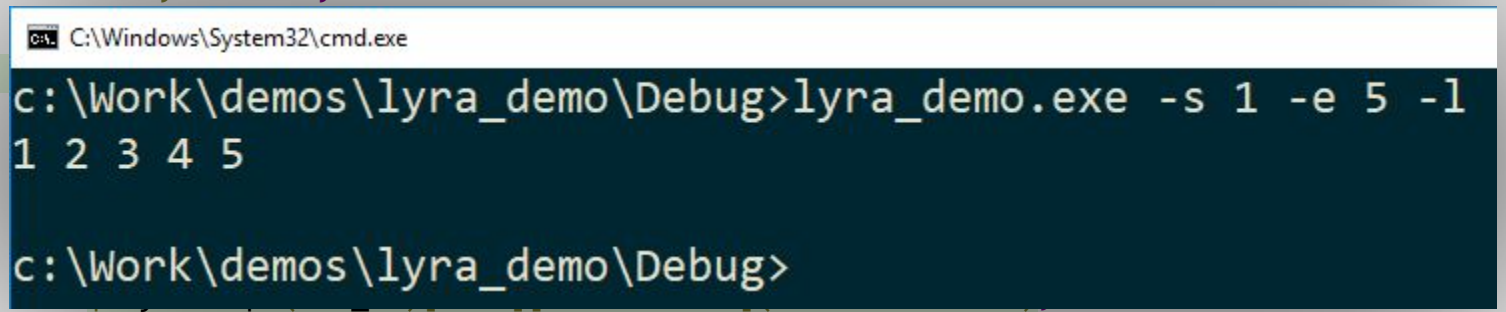
Flag arguments

```
int main(int argc, char** argv)
{
   int start = 0, end = 0;
   auto show_help = false;
   auto add_nl = false;
   auto cli = lyra::help(show_help)
        | lyra::opt(start, "start")["-s"]["--start"]("First number")
        | lyra::opt(end, "end")["-e"]["--end"]("Last number")
        | lyra::opt(add_nl)["-l"]["--newline"]("Add new line");

   auto result = cli.parse({ argc, argv });
   /* ... */

   for (int i = start; i <= end; ++i)
      fmt::print("{} ", i);
   if (add_nl) fmt::print("\n");
}
```

```
C:\Windows\System32\cmd.exe
c:\Work\demos\lyra_demo\Debug>lyra_demo.exe -s 1 -e 5
1 2 3 4 5
c:\Work\demos\lyra_demo\Debug>
```

Flag arguments

```
int main(int argc, char** argv)
{
   int start = 0, end = 0;
   auto
   auto
   auto

   auto result = cli.parse({ argc, argv });
   /* ... */

   for (int i = start; i <= end; ++i)
      fmt::print("{} ", i);
   if (add_nl) fmt::print("\n");
}
```



Flag arguments

```
int main(int ar
{
    int start =
    auto show_he
    auto add_nl
    auto cli = l
               | l
               | l
               | l
               | l

    auto result
    /* ... */

    for (int i =
        fmt::prin
    if (add_nl)
}
```

```
C:\Windows\System32\cmd.exe
c:\Work\demos\lyra_demo\Debug>lyra_demo.exe -h
Usage:
  lyra_demo.exe [-?|-h|--help] [-s|--start] [-e|--end] [-l|--newline]

Options, arguments:
  -?, -h, --help

  Display usage information.

  -s, --start <start>

  Start or sequence

  -e, --end <end>

  End of sequence

  -l, --newline

  Add new line
```

User-defined types arguments

```
enum class verbosity_level
{
   low, normal, debug
};

template<>
struct fmt::formatter<verbosity_level>
{
   template <typename ParseContext>
   constexpr auto parse(ParseContext& ctx) { return ctx.begin(); }

   template <typename FormatContext>
   auto format(verbosity_level const v, FormatContext& ctx)
   {
      /* ... */
   }
};
```

User-defined types arguments

```cpp
std::ostream& operator <<(std::ostream& stream, verbosity_level& level)
{
   switch (level)
   {
   case verbosity_level::low:      stream << "low"; break;
   case verbosity_level::normal:   stream << "normal"; break;
   case verbosity_level::debug:    stream << "debug";  break;
   }

   return stream;
}
```

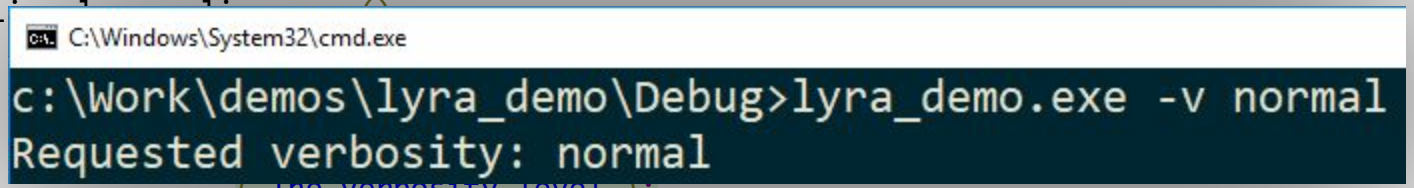
```cpp
std::istream& operator >>(std::istream& stream, verbosity_level& level)
{
   std::string token;
   stream >> token;
   if (token == "low") level = verbosity_level::low;
   else if (token == "normal") level = verbosity_level::normal;
   else if (token == "debug") level = verbosity_level::debug;
   else {
      auto parsed = false;
      try {
         auto n = std::stoi(token);
         if (n >= static_cast<int>(verbosity_level::low) &&
            n <= static_cast<int>(verbosity_level::debug))
         {
            level = static_cast<verbosity_level>(n);
            parsed = true;
         }
      }
      catch (std::exception const&) {}
      if (!parsed)
         throw std::runtime_error("Invalid verbosity level value");
   }

   return stream;
}
```

User-defined types arguments

```
int main(int argc, char** argv)
{
   auto show_help = false;
   auto verbosity = verbosity_level::low;
   auto cli = lyra::cli_parser()
            | lyra::help(show_help)
            | lyra::opt(verbosity, "low|normal|debug")
                      ["-v"]["--verbosity"]
                      ("The verbosity level");

   auto result = cli.parse({ argc, argv });
   /* ... */

   fmt::print("Requested verbosity: {}", verbosity);
}
```

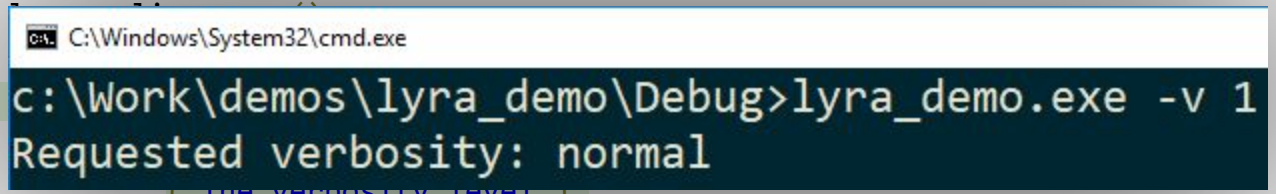
User-defined types arguments

```cpp
int main(int argc, char** argv)
{
    auto show_help = false;
    auto verbosity = verbosity_level::low;
    auto cl
                    ( "The verbosity level" );

    auto result = cli.parse({ argc, argv });
    /* ... */

    fmt::print("Requested verbosity: {}", verbosity);
}
```



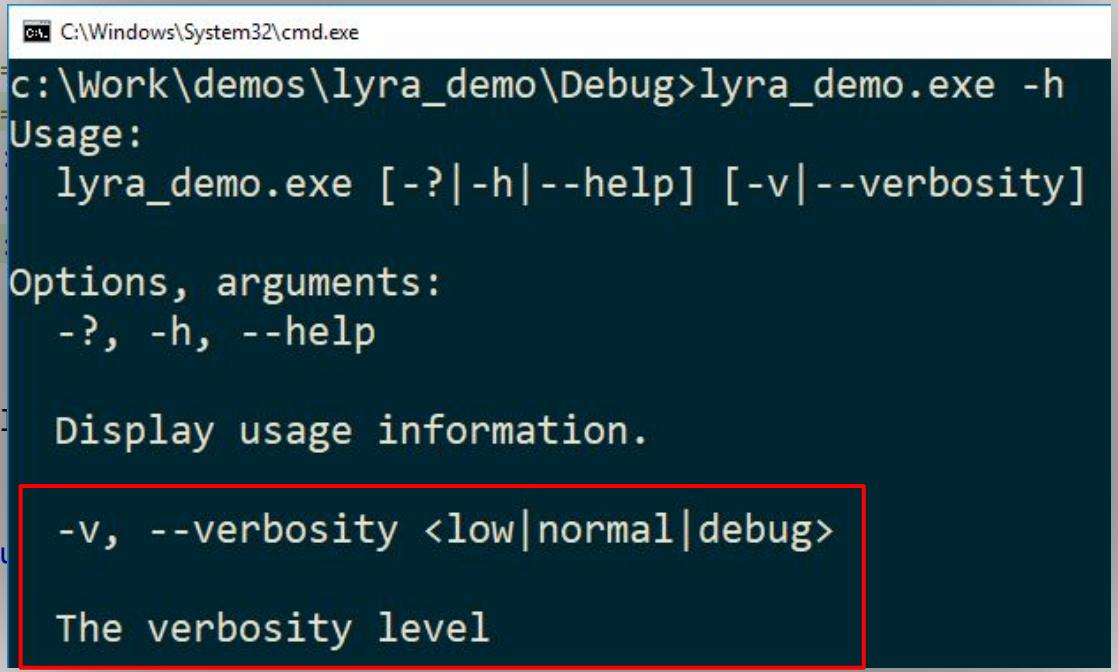
User-defined types arguments

```cpp
int main(int argc, char** argv)
{
    auto show_help = false;
    auto verbosity = verbosity_level::low;
    auto cli =
            |
            |
                    ( "The verbosity level" );

    auto result = cli.parse({ argc, argv });
    /* ... */

    fmt::print("Requested verbosity: {}", verbosity);
}
```



User-defined types arguments

```
int main(int argc, char** argv)
{
    auto show_help =
    auto verbosity =
    auto cli = lyra
             | lyra
             | lyra



    auto result = cl
    /* ... */

    fmt::print("Requ
}
```

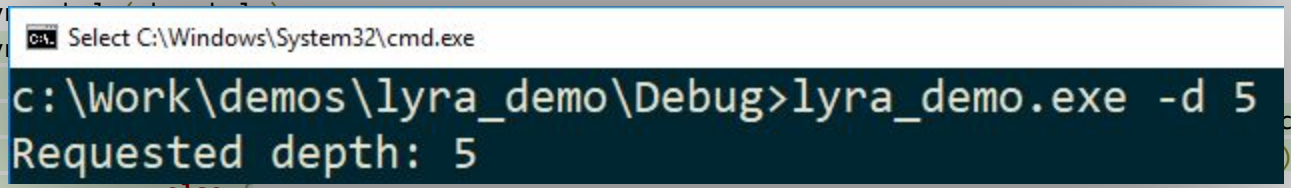


Custom parsing

```cpp
int main(int argc, char** argv)
{
   auto show_help = false;
   auto depth = 0;
   auto cli = lyra::cli_parser()
            | lyra::help(show_help)
            | lyra::opt([&depth](int const d) {
                        if (d < 0 || d > 10)
                           return lyra::parser_result::runtimeError(lyra::parser_result_type::no_match,
                                                                    "Depth must be between 1 and 10");
                        else {
                           depth = d;
                           return lyra::parser_result::ok(lyra::parser_result_type::matched);
                        }
                     },"depth")
                     ["-d"]["--depth"] ("Depth of parsing (1 to 10)");

   auto result = cli.parse({ argc, argv });
   /* ... */

   fmt::print("Requested depth: {}", depth);
}
```

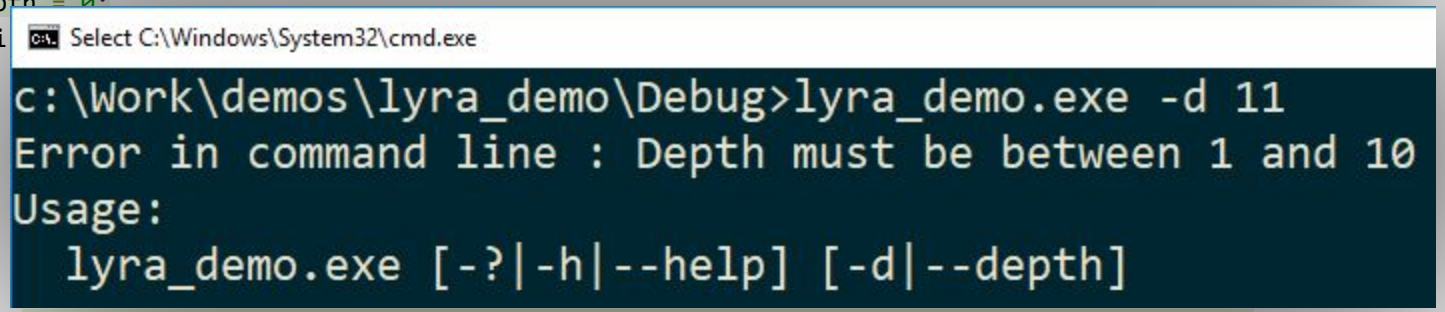
Custom parsing

```
int main(int argc, char** argv)
{
   auto show_help = false;
   auto depth = 0;
   auto cli = lyra::cli_parser()
            | ly
            | ly
                                                                    ch,
                                                                    );
                        else {
                           depth = d;
                           return lyra::parser_result::ok(lyra::parser_result_type::matched);
                        }
                     },"depth")
                     ["-d"]["--depth"] ("Depth of parsing (1 to 10)");

   auto result = cli.parse({ argc, argv });
   /* ... */

   fmt::print("Requested depth: {}", depth);
}
```



Custom parsing

```
int main(int argc, char** argv)
{
   auto show_help = false;
   auto depth = 0;
   auto cli
                  return lyra::parser_result::ok(lyra::parser_result_type::matched);
               }
            },"depth")
            ["-d"]["--depth"] ("Depth of parsing (1 to 10)");

   auto result = cli.parse({ argc, argv });
   /* ... */

   fmt::print("Requested depth: {}", depth);
}
```



Custom parsing

```
int main(int argc, char** argv)
{
   auto show_help = false;
   auto depth = 0;
   auto cli = lyra::cli_pa
            | lyra::help(s
            | lyra::opt([&
                                                                    e::no_match,
                                                                    1 and 10");
                                                                    );
                    },
                    ["
   auto result = cli.parse
   /* ... */

   fmt::print("Requested depth: {}", depth);
}
```

```
C:\Windows\System32\cmd.exe
c:\Work\demos\lyra_demo\Debug>lyra_demo.exe -h
Usage:
  lyra_demo.exe [-?|-h|--help] [-d|--depth]

Options, arguments:
  -?, -h, --help

  Display usage information.

  -d, --depth <depth>

  Depth of parsing (1 to 10)
```

Catch2

https://github.com/catchorg/Catch2

catchorg / **Catch2**

Sponsor | Watch 436 | Star 10.8k | Fork 1.8k

Code | Issues 211 | Pull requests 29 | Actions | Projects 0 | Security | Insights

A modern, C++-native, header-only, test framework for unit-tests, TDD and BDD - using C++11, C++14, C++17 and later (or C++03 on the Catch1.x branch) https://discord.gg/4CWS9zD

c-plus-plus | cpp98 | cpp11 | testing | test-framework | tdd | bdd | header-only | single-file | no-dependencies | framework

3,130 commits | 9 branches | 0 packages | 95 releases | 205 contributors | BSL-1.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

horenmar Specialize CATCH_TRAP() for iOS + thumb instruction set combo ... Latest commit 81d52c4 8 days ago

.conan	Fix typos identified by codespell.	11 months ago
.github	Add a FUNDING file	9 months ago
CMake	Fix typos identified by codespell.	11 months ago
artwork	Optimize in-repo PNGs	6 months ago
contrib	Fix CTest command issue with square brackets	7 months ago

catchorg / **Catch2**

Sponsor | Watch 436 | Star 10.8k | Fork 1.8k

Code | Issues 211 | Pull requests 29 | Actions | Projects 0 | Security | Insights

A modern, C++-native, header-only, test framework for unit-tests, TDD and BDD - using C++11, C++14, C++17 and later (or C++03 on the Catch1.x branch) https://discord.gg/4CWS9zD

c-plus-plus | cpp98 | cpp11 | testing | test-framework | tdd | bdd | header-only | single-file | no-dependencies | framework

3,130 commits | **9** branches | **0** packages | **95** releases | **205** contributors | BSL-1.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

horenmar Specialize CATCH_TRAP() for iOS + thumb instruction set combo ... Latest commit 81d52c4 8 days ago

.conan	Fix typos identified by codespell.	11 months ago
.github	Add a FUNDING file	9 months ago
CMake	Fix typos identified by codespell.	11 months ago
artwork	Optimize in-repo PNGs	6 months ago
contrib	Fix CTest command issue with square brackets	7 months ago

Intro

- Test framework for unit-tests, TDD and BDD
- Open-source, cross-platform
- C++11 and beyond
- Single header
- Multiple fixture styles
- Also supports Objective-C

Fizzbuzz

```cpp
#include <string>

std::string fizzbuzz(int const number)
{
   if (number != 0)
   {
      auto m3 = number % 3;
      auto m5 = number % 5;
      if (!m5 && !m3) { return "fizzbuzz"; }
      else if (!m5) { return "buzz"; }
      else if (!m3) { return "fizz"; }
   }

   return std::to_string(number);
}
```

```
 0 -> 0
 1 -> 1
 2 -> 2
 3 -> fizz
 4 -> 4
 5 -> buzz
 6 -> fizz
 7 -> 7
 8 -> 8
 9 -> fizz
10 -> buzz
...
15 -> fizzbuzz
...
```

Anatomy of a test

```
#include "fizzbuzz.h"

#define CATCH_CONFIG_MAIN
#include "catch.hpp"




TEST_CASE("Test with zero", "[classic]")
{
   REQUIRE(fizzbuzz(0) == "0");
}
```

Anatomy of a test

```
#include "fizzbuzz.h"

#define CATCH_CONFIG_MAIN
#include "catch.hpp"


TEST_CASE("Test with zero", "[classic]")
{
    REQUIRE(fizzbuzz(0) == "0");
}
```

Instructs to define the function `main()`

Header file

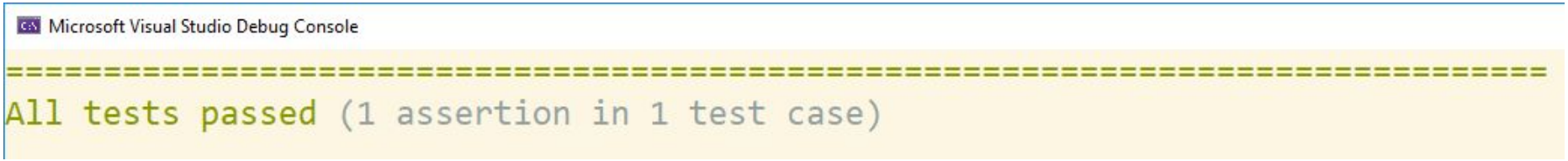
Macro that defines a test function

Name of the test case

Tags

Assert

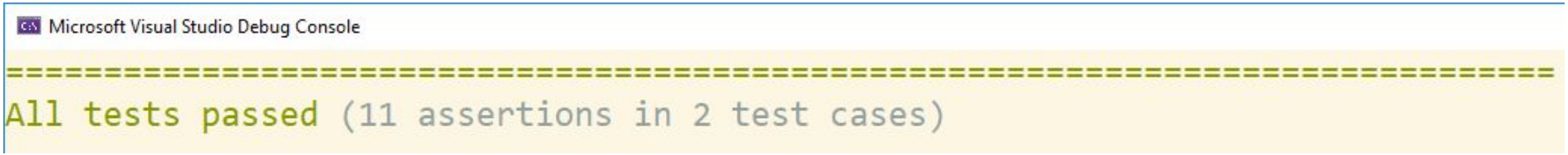
Result


Microsoft Visual Studio Debug Console
==========
All tests passed (1 assertion in 1 test case)

More tests

```
TEST_CASE("Test all up to 10", "[classic]")
{
   REQUIRE(fizzbuzz(1) == "1");
   REQUIRE(fizzbuzz(2) == "2");
   REQUIRE(fizzbuzz(3) == "fizz");
   REQUIRE(fizzbuzz(4) == "4");
   REQUIRE(fizzbuzz(5) == "buzz");
   REQUIRE(fizzbuzz(6) == "fizz");
   REQUIRE(fizzbuzz(7) == "7");
   REQUIRE(fizzbuzz(8) == "8");
   REQUIRE(fizzbuzz(9) == "fizz");
   REQUIRE(fizzbuzz(10) == "buzz");
}
```

Result


Microsoft Visual Studio Debug Console
==========
All tests passed (11 assertions in 2 test cases)

More tests

```
TEST_CASE("Test all up to 10", "[classic]")
{
    REQUIRE(fizzbuzz(1) == "1");
    REQUIRE(fizzbuzz(2) == "2");
    REQUIRE(fizzbuzz(3) == "fizz");
    REQUIRE(fizzbuzz(4) == "fizz");
    REQUIRE(fizzbuzz(5) == "buzz");
    REQUIRE(fizzbuzz(6) == "fizz");
    REQUIRE(fizzbuzz(7) == "7");
    REQUIRE(fizzbuzz(8) == "8");
    REQUIRE(fizzbuzz(9) == "fizz");
    REQUIRE(fizzbuzz(10) == "buzz");
}
```

Result

Microsoft Visual Studio Debug Console

```
~~~
```

More tests

```
TEST_CASE("Test positives", "[classic]")
{
   SECTION("Test all multiples of 3 only up to 100") {
      for (int i = 3; i <= 100; i+=3) {
         if (i % 5) REQUIRE(fizzbuzz(i) == "fizz");
      }
   }

   SECTION("Test all multiples of 5 only up to 100") {
      for (int i = 5; i <= 100; i += 5) {
         if (i % 3) REQUIRE(fizzbuzz(i) == "buzz");
      }
   }

   SECTION("Test all multiples of 3 and 5 ") {
      for (int i = 15; i <= 100; i += 15) {
         REQUIRE(fizzbuzz(i) == "fizzbuzz");
      }
   }
}
```

Sections

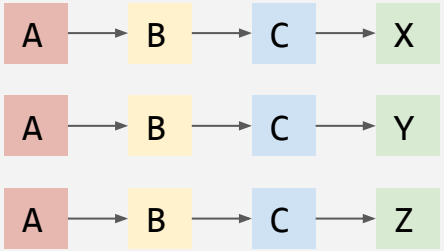
- Inside a test case
- For each section the test case is executed from the start
- Sections can be nested (no limit)
- Each leaf section is executed exactly once on a separate path of execution from all other leafs
- No explicit fixtures (setup/teardown, constructor/destructor)

Section execution

```
TEST_CASE( "fixtures", "[demo]" ) {

    // init

    SECTION( "A" ) {
       SECTION( "B" ) {
          SECTION( "C" ) {
             SECTION( "X" ) {
             }
             SECTION( "Y" ) {
             }
             SECTION( "Z" ) {
             }
          }
       }
    }
}
```

Section execution

```
TEST_CASE( "fixtures", "[demo]" ) {

    // init

    SECTION( "A" ) {
        SECTION( "B" ) {
            SECTION( "C" ) {
                SECTION( "X" ) {
                }
                SECTION( "Y" ) {
                }
                SECTION( "Z" ) {
                }
            }
        }
    }
}
```



Fixtures

```
TEST_CASE( "vectors can be sized and resized", "[vector]" )
{
    std::vector<int> v( 5 );
    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }

    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );
    }
}
```

Fixtures

```
TEST_CASE( "vectors can be sized and resized", "[vector]" )
{
    std::vector<int> v( 5 );
    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }

    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );
    }
}
```

1st path

Fixtures

```
TEST_CASE( "vectors can be sized and resized", "[vector]" )
{
    std::vector<int> v( 5 );
    REQUIRE( v.size() == 5 );
    REQUIRE( v.capacity() >= 5 );

    SECTION( "resizing bigger changes size and capacity" ) {
        v.resize( 10 );

        REQUIRE( v.size() == 10 );
        REQUIRE( v.capacity() >= 10 );
    }

    SECTION( "resizing smaller changes size but not capacity" ) {
        v.resize( 0 );

        REQUIRE( v.size() == 0 );
        REQUIRE( v.capacity() >= 5 );
    }
}
```

2nd path

BDD tests

```
SCENARIO("BDD test with zero", "[bdd]")
{
   WHEN("The number is 0") {
      THEN("The result is 0") {
         REQUIRE(fizzbuzz(0) == "0");
      }
   }
}
```

BDD tests

```
SCENARIO("BDD test any number", "[bdd]") {
   GIVEN("Any positive number") {
      WHEN("The number is 1") {
         THEN("The result is 1") {
            REQUIRE(fizzbuzz(1) == "1");
         }
      }

      WHEN("The number is 3") {
         THEN("The result is fizz") {
            REQUIRE(fizzbuzz(3) == "fizz");
         }
      }

      WHEN("The number is 5") {
         THEN("The result is buzz") {
            REQUIRE(fizzbuzz(5) == "buzz");
         }
      }
   }
}
```

BDD tests

```
#define SCENARIO( ... )  TEST_CASE( "Scenario: " __VA_ARGS__ )
#define GIVEN( desc )    SECTION( std::string("   Given: ") + desc )
#define WHEN( desc )     SECTION( std::string("    When: ") + desc )
#define AND_WHEN( desc ) SECTION( std::string("And when: ") + desc )
#define THEN( desc )     SECTION( std::string("    Then: ") + desc )
#define AND_THEN( desc ) SECTION( std::string("     And: ") + desc )
```

Assertions

Execution stops	Execution continues	Description
`REQUIRE(expr)`	`CHECK(expr)`	Expects that expr is evaluated to true.
`REQUIRE_FALSE(expr)`	`CHECK_FALSE(expr)`	Expects that expr is evaluated to false.
`REQUIRE_NOTHROW(expr)`	`CHECK_NOTHROW(expr)`	Expects that no exception is thrown by the evaluation of expr
`REQUIRE_THROWS(expr)`	`CHECK_THROWS(expr)`	Expects that an exception is thrown during the evaluation of expr
`REQUIRE_THROW_AS(expr, type)`	`CHECK_THROW_AS(expr, type)`	Expects that an exception of type is thrown during the evaluation of expr
`REQUIRE_THROWS_WITH(expr, matcher)`	`CHECK_THROWS_WITH(expr, matcher)`	Expects that an exception is thrown during the evaluation of expr that when converted to string matches the provided string or string matcher.
`REQUIRE_THROWS_MATCHES(expr, type, matcher)`	`CHECK_THROWS_MATCHES(expr, type, matcher)`	Expects that exception of type is thrown and matches the provided matcher.
`REQUIRE_THAT(expr, matcher)`	`CHECK_THAT(expr, matcher)`	Expects that the evaluation of expr matches the provided matcher(s).

Logging macros

Macro	Description
INFO(msg)	Prints message in the output only if no failure occurs in its scope.
UNSCOPED_INFO(msg)	Similar but reported by the failure of the first following assertion.
WARN(msg)	The message is always reported but does not fail the test.
FAIL(msg)	The message is reported and the test case fails.
FAIL_CHECK(msg)	As fail but does not abort the test.
CAPTURE(expr1, expr2, expr3, ...)	Prints the variable/expression and its value at the time of capture.

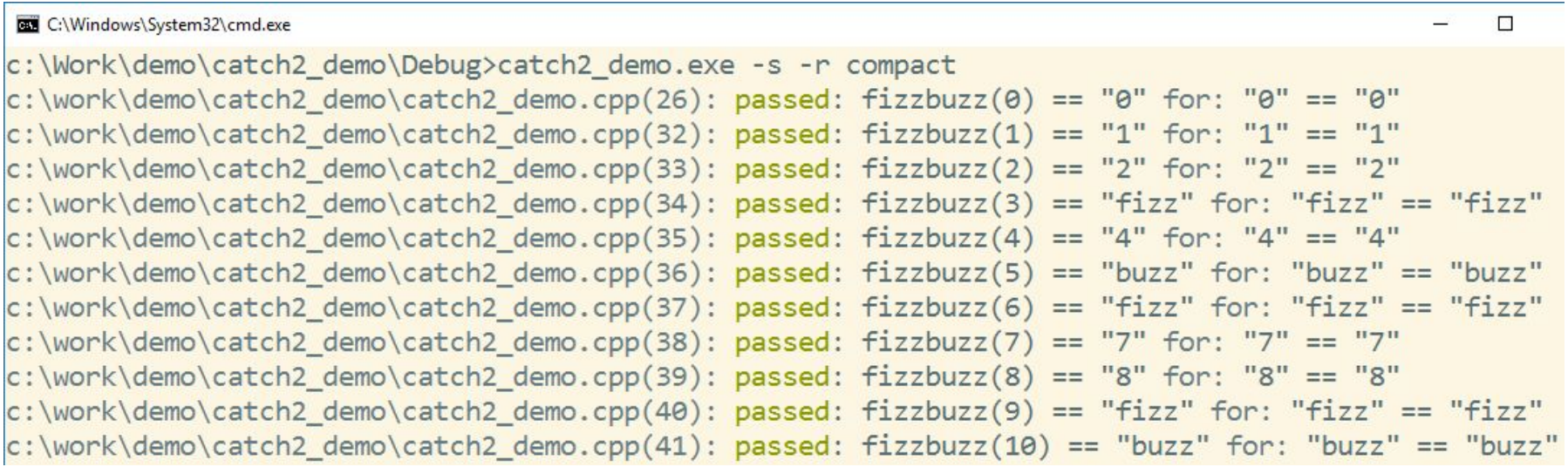
Command line options

Show results for successful tests (`-s`)

C:\Windows\System32\cmd.exe

```
c:\work\demo\catch2_demo\catch2_demo.cpp(71):
PASSED:
  REQUIRE( fizzbuzz(-7) == "-7" )
with expansion:
  "-7" == "-7"

c:\work\demo\catch2_demo\catch2_demo.cpp(72):
PASSED:
  REQUIRE( fizzbuzz(-8) == "-8" )
with expansion:
  "-8" == "-8"

c:\work\demo\catch2_demo\catch2_demo.cpp(73):
PASSED:
  REQUIRE( fizzbuzz(-9) == "fizz" )
with expansion:
  "fizz" == "fizz"

c:\work\demo\catch2_demo\catch2_demo.cpp(74):
PASSED:
  REQUIRE( fizzbuzz(-10) == "buzz" )
with expansion:
  "buzz" == "buzz"

===============================================================
All tests passed (68 assertions in 3 test cases)
```

Show compact results (`-s -r -compact`)

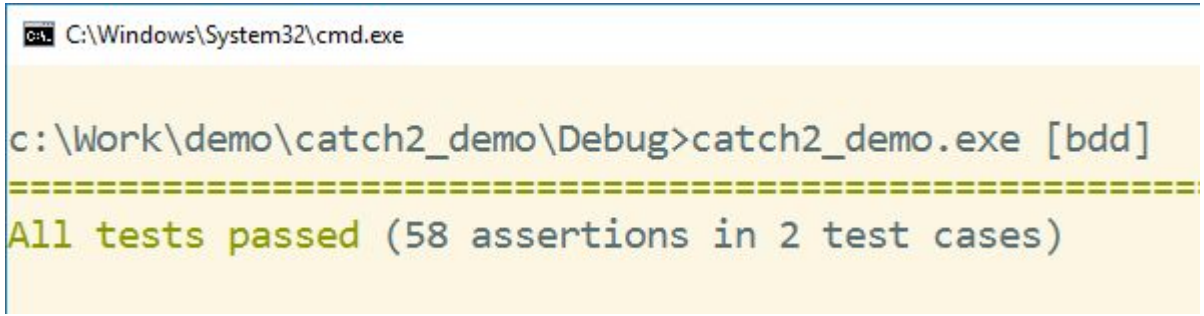


Show results as JUnit XML Report ANT (`-junit`)

C:\Windows\System32\cmd.exe

```
c:\Work\demo\catch2_demo\Debug>catch2_demo.exe -r junit
<?xml version="1.0" encoding="UTF-8"?>
<testsuites>
  <testsuite name="catch2_demo.exe" errors="0" failures="0" tests="68" hostname="tbd"
018-03-29T11:52:05Z">
    <system-out/>
    <system-err/>
  </testsuite>
</testsuites>
```

Execute tests with a tag



More features

- Machers
- Data generators
- Event listener
- Reporters
- String conversions
- etc.

nlohmann/json

https://github.com/nlohmann/json

nlohmann / json

Sponsor | Watch 627 | Unstar 17.5k | Fork 3k

Code | Issues 31 | Pull requests 7 | Actions | Projects 1 | Security | Insights

JSON for Modern C++ https://nlohmann.github.io/json/

json json-pointer json-patch cbor msgpack header-only stl-containers json-serialization json-parser rfc-6901 rfc-6902 rfc-7159 rfc-7049 json-diff messagepack rfc-8259 ubjson json-merge-patch bson

3,249 commits | 6 branches | 0 packages | 31 releases | 152 contributors | MIT

Branch: develop | New pull request | Create new file | Upload files | Find file | Clone or download

nlohmann Merge pull request #1923 from Coeur/typos ... Latest commit 973c52d 15 days ago

.circleci	add version output	4 months ago
.github	Some typos	16 days ago
benchmarks	Use C++11 features supported by CMake 3.1.	13 months ago
cmake	cmake: fix package config to deal with versioning and namespaces	17 months ago
doc	Some typos	16 days ago
include/nlohmann	Some typos	16 days ago

Overview

- Open source, cross platform, for C++11
- Single header
- Trivial integration
- Json literals
- Conversion from STL containers
- Easy serialization and deserialization for user-defined types

Example

```
{
  "pi": 3.141,
  "happy": true,
  "name": "Niels",
  "nothing": null,
  "answer": {
    "everything": 42
  },
  "list": [1, 0, 2],
  "object": {
    "currency": "USD",
    "value": 42.99
  }
}
```

```
// create an empty structure (null)
json j;

// add a number that is stored as double (note the implicit conversion of j to an object)
j["pi"] = 3.141;

// add a Boolean that is stored as bool
j["happy"] = true;

// add a string that is stored as std::string
j["name"] = "Niels";

// add another null object by passing nullptr
j["nothing"] = nullptr;

// add an object inside the object
j["answer"]["everything"] = 42;

// add an array that is stored as std::vector (using an initializer list)
j["list"] = { 1, 0, 2 };

// add another object (using an initializer list of pairs)
j["object"] = { {"currency", "USD"}, {"value", 42.99} };
```

Source: https://github.com/nlohmann/json

Example

```
{
  "pi": 3.141,
  "happy": true,
  "name": "Niels",
  "nothing": null,
  "answer": {
    "everything": 42
  },
  "list": [1, 0, 2],
  "object": {
    "currency": "USD",
    "value": 42.99
  }
}
```

```
// instead, you could also write
json j2 = {
  {"pi", 3.141},
  {"happy", true},
  {"name", "Niels"},
  {"nothing", nullptr},
  {"answer", {
    {"everything", 42}
  }},
  {"list", {1, 0, 2}},
  {"object", {
    {"currency", "USD"},
    {"value", 42.99}
  }}
};
```

Source: https://github.com/nlohmann/json

Example: conversion from STL containers

```
std::vector<int> c_vector {1, 2, 3, 4};
json j_vec(c_vector);
// [1, 2, 3, 4]

std::array<unsigned long, 4> c_array {{1, 2, 3, 4}};
json j_array(c_array);
// [1, 2, 3, 4]

std::map<std::string, int> c_map { {"one", 1}, {"two", 2}, {"three", 3} };
json j_map(c_map);
// {"one": 1, "three": 3, "two": 2 }
```

Example: implicit conversions

```
// strings
std::string s1 = "Hello, world!";
json js = s1;
auto s2 = js.get<std::string>();

// Booleans
bool b1 = true;
json jb = b1;
auto b2 = jb.get<bool>();

// numbers
int i = 42;
json jn = i;
auto f = jn.get<double>();
```

Example: user-defined types

```cpp
namespace ns {
  struct person {
    std::string name;
    std::string address;
    int age;
  };
}
```

```cpp
using nlohmann::json;

namespace ns {
  void to_json(json& j, const person& p) {
    j = json{ {"name", p.name}, {"address", p.address},
              {"age", p.age} };
  }

  void from_json(const json& j, person& p) {
    j.at("name").get_to(p.name);
    j.at("address").get_to(p.address);
    j.at("age").get_to(p.age);
  }
}
```

Example: user-defined types

```cpp
namespace ns {
  struct person {
    std::string name;
    std::string address;
    int age;
  };
}
```

```cpp
// create a person
ns::person p {"Ned Flanders", "744 Evergreen Terrace", 60};

// conversion: person -> json
json j = p;

std::cout << j << std::endl;
// {"address":"744 Evergreen Terrace","age":60,"name":"Ned
Flanders"}

// conversion: json -> person
auto p2 = j.get<ns::person>();

// that's it
assert(p == p2);
```

Thank you

Q&A

One more thing...

Use east const!

```
int const a = 42;
```

Join the revolution!

Petition for const consistency
http://slashslash.info/petition/