

`std::format` *and* `{fmt}` – *an overview*

St. Louis C++ Meetup

December 4, 2019

John Fultz

What is `{fmt}`?

- The `fmt` library, styled `{fmt}`, authored largely by Victor Zverovich
- Solves the problem of formatting text, a la `sprintf`, et. al
- Formats all basic types consistently cross-platform
- Extensible for user types

Goals of `{fmt}`

- Safety
 - Type checking
 - Argument counting
 - Memory safe
- Compact generated code
- Portability
 - Versions 5.x and 6.x work on any C++11 compiler
 - Version 4.x continues to be maintained at C++98
 - No external dependencies
- Reuse tested formatting conventions from Python/Rust
- Speed
 - Benchmarks among the fastest for non-locale

The competition – printf

- Widely available
- Generally fast
- No safety at all
- No extensibility
- C-specific

The competition – iostream

- Widely available
 - Safe
 - Extensible
 - Extremely verbose – “chevron hell”
 - E.g.,

```
std::cout << std::setprecision(2) << std::fixed << 1.23456 << "\n";
```
- VS.
- ```
printf("%.2f\n", 1.23456);
```
- Painfully slow number conversions

# The competition – Boost Format

- Powerful
- Safe
- Extensible
  
- Slow performance
- Bloated generated code

# Using `format()`

- No compiler I've found has it in the C++20 feature set, yet.
- The `fmt` library, also styled as `{fmt}`, is available under a very liberal license (MIT-style for open source, effectively unlimited for redistribution of compiled objects)
- Available in `vcpkg`, HomeBrew, GitHub, and others
- Also available in the list of libraries on [godbolt.org](https://godbolt.org)
- Compiled library + headers; also can be used as header-only









# An even easier {fmt} version, but not coming in C++20

```
#include <iostream>
#include "fmt/format.h"

int main()
{
 fmt::print("Hello {}. ", "world");
 return 0;
}
```

# Demo code

- 01\_hello\_world.cpp – {fmt} hello world
- 02\_hello\_word\_std.cpp – C++20 hello world
- 03\_hello\_world\_fmt.cpp – {fmt} hello world minus iostream
- 04\_basic\_types.cpp – support for basic C++ types
- 05\_format\_string.cpp – various format string specs
- 06\_output\_iterator.cpp – output to an iterator instead of `std::string`
- 07\_formatter\_specialization.cpp – add support for custom types
- 08\_error\_handling.cpp – catching type errors

# Benchmarks – performance

Times run over a variety of format types, macOS

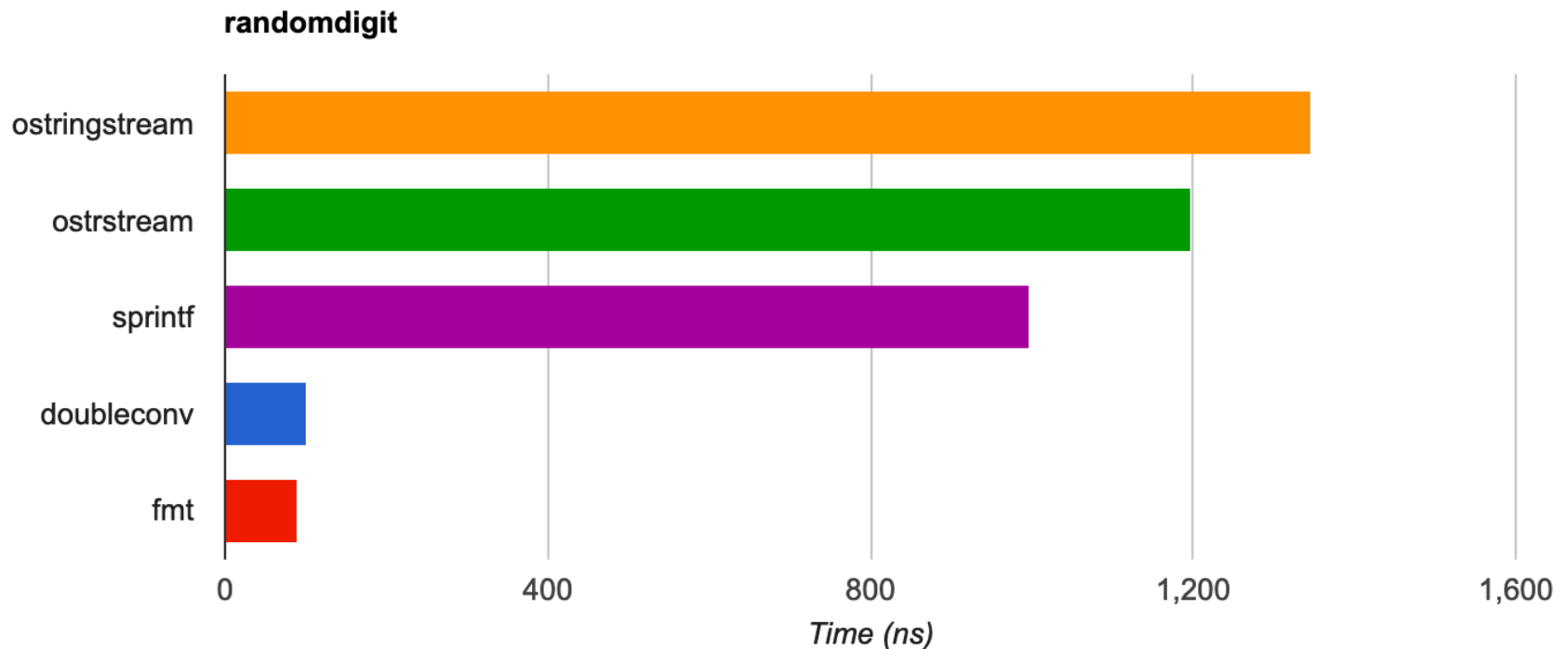
Source: {fmt} readme

| Library           | Method        | Run Time, s |
|-------------------|---------------|-------------|
| libc              | printf        | 1.03        |
| libc++            | std::ostream  | 2.98        |
| {fmt} 4de41a      | fmt::print    | 0.76        |
| Boost Format 1.67 | boost::format | 7.24        |
| Folly Format      | folly::format | 2.23        |

# Benchmarks – performance

Milo Yip's d2a Benchmark – <https://github.com/miloyip/dtoa-benchmark>

Source: {fmt} readme



# Benchmarks – compiler, -03

Times run over a variety of format types, macOS

Source: `{fmt}` readme

| Method        | Compile Time, s | Executable size, KiB | Stripped size, KiB |
|---------------|-----------------|----------------------|--------------------|
| printf        | 2.6             | 29                   | 26                 |
| printf+string | 16.4            | 29                   | 26                 |
| iostreams     | 31.1            | 59                   | 55                 |
| {fmt}         | 19.0            | 37                   | 34                 |
| Boost Format  | 91.9            | 226                  | 203                |
| Folly Format  | 115.7           | 101                  | 88                 |

# Benchmarks – compiler, -O0

Times run over a variety of format types, macOS

Source: {fmt} readme

| Method        | Compile Time, s | Executable size, KiB | Stripped size, KiB |
|---------------|-----------------|----------------------|--------------------|
| printf        | 2.2             | 33                   | 30                 |
| printf+string | 16.0            | 33                   | 30                 |
| iostreams     | 28.3            | 56                   | 52                 |
| {fmt}         | 18.2            | 59                   | 50                 |
| Boost Format  | 54.1            | 365                  | 303                |
| Folly Format  | 79.9            | 445                  | 430                |



# The chrono question

- C++20 includes significantly updated chrono formatting and scanning functionality
- <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0355r7.html>
- The chrono proposal interacts with the formatting proposal by adding a chrono-specific `std::formatter`
- [https://en.cppreference.com/w/cpp/chrono/system\\_clock/formatter](https://en.cppreference.com/w/cpp/chrono/system_clock/formatter)
- <https://en.cppreference.com/w/cpp/utility/format/formatter>
- `{fmt}` has time formatting in “`fmt/chrono.h`”, but it’s not anything like the C++20 proposal
- <https://isocpp.org/files/papers/P1361R1.pdf>

# What we aren't getting in C++20

- `fmt::print` – a printing function

- Compile-time type checking

```
std::string s = format(fmt("{:d}"), "foo");
```

- Literal support

```
using namespace fmt::literals;
```

```
std::string message = "The answer is {}"_format(42);
```

- Named positions

```
auto msg = fmt::format("{key2}: {key1}", fmt::arg("key1", 1),
 fmt::arg("key2", "abc"));
```

# Links

- `{fmt}` info
  - <https://fmt.dev/>
  - <https://github.com/fmtlib/fmt>
  - [https://youtu.be/ptba\\_AqFYCM](https://youtu.be/ptba_AqFYCM) (Zverovich's CppCon 2017 talk)
- More on integer and FP to text conversions
  - <https://www.zverovich.net/2019/02/11/formatting-floating-point-numbers.html>
  - <https://github.com/miloyip/itoa-benchmark>
  - <https://github.com/miloyip/dtoa-benchmark>
  - [https://youtu.be/4P\\_kbF0EbZM](https://youtu.be/4P_kbF0EbZM) (CppCon 2019 talk on `<charconv>`)