6-format

June 3, 2024

1 The awaited return of the format string

1.1 Motivation

C++ makes available the I/O functions from the C ecosystem: `printf`, `scanf`, and their many variations.

On top of it, C++ provides an input/output library, highly object-oriented, therefore highly extendable (typically for new user-defined classes). It also has better type safety.

Yet, the new library is often considered cumbersome and verbose. It lacks the very appreciated conciseness of the good old-fashioned C functions.

1.2 Idea

The new C++20 `std::format` function aims to fill this gap: provide a concise way to pack many values into a `std::string`, which can then be feeded into the usual C++ streams.

In the format string, placeholders to be filled are materialized with a pair of curly braces (very python-like…):

```
#include <iostream>
#include <format>

int main()
 {
  std::cout<<std::format("The answer is {}.",3.14)<<std::endl ;
 }
```

1.3 The {fmt} library

Only the last versions of the compilers start to support the new text formatting features. Meanwhile, one can use the {fmt} library, which was the base of the C++20 proposal.

[51]:
```
%%file tmp.format.cpp

#include <iostream>
#include <fmt/core.h>

int main()
 {
```

```
 std::cout<<fmt::format("The answer is {}.",42)<<std::endl ;
}
```

```
Overwriting tmp.format.cpp
```

[52]:
```
!rm -f tmp.format.exe && g++ -std=c++20 tmp.format.cpp -lfmt -o tmp.format.exe
```

[53]:
```
!./tmp.format.exe
```

```
The answer is 42.
```

1.4 Reordering the values

If there are several placeholders in the format string, they are filled one by one with the other arguments. Yet, one can add numbers to reorder or reuse the values.

[54]:
```
%%file tmp.format.cpp

#include <iostream>
#include <fmt/core.h>

struct Vector3d { double x, y, z ; };

int main()
 {
  Vector3d v { 1./2., 2./2., 2./3. } ;
  std::cout<<fmt::format("{1}{0}{2}{0}{3}",'|',v.x,v.y,v.z)<<std::endl ;
 }
```

```
Overwriting tmp.format.cpp
```

[55]:
```
!rm -f tmp.format.exe && g++ -std=c++20 tmp.format.cpp -lfmt -o tmp.format.exe
```

[56]:
```
!./tmp.format.exe
```

```
0.5|1|0.6666666666666666
```

1.5 Formatting modifiers for builtin types

Within each placeholder, one can given additional printf-like options, after a : separator. See cppreference.com for a complete documentation.

[57]:
```
%%file tmp.format.cpp

#include <iostream>
#include <fmt/core.h>

struct Vector3d { double x, y, z ; };

int main()
```

```
 {
  Vector3d v { 1./2., 2./2., 2./3. } ;
  std::cout<<fmt::format("vector: {1:.2f}{0}{2:.2f}{0}{3:.2f}",'|',v.x,v.y,v.
 ↪z)<<std::endl ;
 }
```

```
Overwriting tmp.format.cpp
```

[58]:
```
!rm -f tmp.format.exe && g++ -std=c++20 tmp.format.cpp -lfmt -o tmp.format.exe
```

[59]:
```
!./tmp.format.exe
```

```
vector: 0.50|1.00|0.67
```

1.6 Formatting user-defined types

One can override the usual C++ IO streams, and it will be

[85]:
```
%%file tmp.format.cpp

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

struct Vector3d { double x, y, z ; } ;

template <>
struct fmt::formatter<Vector3d>
 {
  char sep ;
  constexpr auto parse(format_parse_context& ctx)
   {
    auto it = ctx.begin(), end = ctx.end();
    if (it != end && *it != '}') sep = *it++ ;
    while (it != end && *it != '}') it++ ;
    return it;
   }

  // Formats the point p using the parsed format specification (presentation)
  // stored in this formatter.
  template <typename FormatContext>
  auto format( const Vector3d & v, FormatContext & ctx ) const
   {
    return fmt::format_to(ctx.out(),"vector: {1:.2f}{0}{2:.2f}{0}{3:.2f}",sep,v.
 ↪x,v.y,v.z) ;
   }
};

int main()
```

```
{
 Vector3d v { 1./2., 2./2., 2./3. } ;
 std::string txt = fmt::format("{:|}",v) ;
 std::cout<<txt<<std::endl ;
}
```

```
Overwriting tmp.format.cpp
```

[86]:
```
!rm -f tmp.format.exe && g++ -std=c++20 tmp.format.cpp -lfmt -o tmp.format.exe
```

[87]:
```
!./tmp.format.exe
```

```
vector: 0.50|1.00|0.67
```

1.7 The {fmt} library vs the C++20 standard library

As of september 2022, *{fmt}* implements nearly all of the C++20 formatting library with the following differences: - Names are defined in the `fmt` namespace instead of `std` to avoid collisions. - Most C++20 chrono types are not supported yet.

There are some nice extra features: - `fmt::print()` as a substitute for `std::cout`, - colored output with foreground and background modifiers, - built-in support for formatting containers, - named arguments, - for a user-defined type `T`, a way to make `operator<<(std::ostream &, T const &)` reusable for `fmt::format`.

1.8 To remember

Thanks to variadic templates and compile-time parsing of the format string, the function `std::format`, - is checking that the types of its arguments are the ones expected, - is faster than `std::printf` or `std::ostream`, - can be extended for user-defined types.

2 Questions ?

3 Availability

- MSVC 16.10 is the only compiler that fully supports `std::format`.
- Clang 14 (with libc++14) also has almost full support.
- GCC 12 doesn't support it yet.
- Meanwhile, one can use the {fmt} library, which was the base of the C++20 proposal.

4 Exercise

In the code below, we would like to choose the number of displayed digits at runtime. Modify the specialized `fmt::formatter<Vector3d>` so that the value given in `main()` (3) is taken into account. Trick: as already done, the first step is to make `fmt::formatter<Vector3d>` inherit from `fmt::formatter<double>`.

[82]:
```
%%file tmp.format.cpp

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

struct Vector3d { double x, y, z ; };

template <>
struct fmt::formatter<Vector3d> : formatter<double>
 {
  char sep ;
  constexpr auto parse(format_parse_context& ctx)
   {
    auto it = ctx.begin(), end = ctx.end();
    // HERE YOU SHOULD EXTRACT THE SEPARATOR,
    // THEN CALL formatter<double>::parse(ctx)
    // SO TO NOT DISTURB THE LATTER WITH THE
    // SEPARATOR LETTER, YOU WILL NEED
    // fmt::format_parse_context::advance_to()
   }

  // Formats the point p using the parsed format specification (presentation)
  // stored in this formatter.
  template <typename FormatContext>
  auto format( Vector3d const & v, FormatContext & ctx ) const -> decltype(ctx.
 ↪out())
   {
    format_to(ctx.out(),"vector: ") ;
    // HERE YOU SHOULD MIX CALLS TO fmt::format_to()
    // AND INHERITED formatter<double>::format()
    return ctx.out() ;
   }
};

int main()
 {
  Vector3d v { 1./2., 2./2., 2./3. } ;
  std::string txt = fmt::format("{:|.3f}",v) ;
  std::cout<<txt<<std::endl ;
 }
```

```
Overwriting tmp.format.cpp
```

[83]:
```
!rm -f tmp.format.exe && g++ -std=c++20 tmp.format.cpp -lfmt -o tmp.format.exe
```

[84]:
```
!./tmp.format.exe
```

```
vector: 0.500|1.000|0.667
```

5 Sources

- cppreference.com
- Modernes C++
- Madrid C/C++