

C'est 2 - A Standard C++ Library with constexpr Extensions

Tools and Libraries for Compile-time Software Engineering
HiPEAC Conference 2024
Munich, Germany

Paul Keir ¹
Joel FALCOU ²

¹School of Computing, Engineering & Physical Sciences
University of the West of Scotland, Paisley, UK

²Le Laboratoire Interdisciplinaire des Sciences du Numérique (LISN)
Université Paris-Saclay, Paris, France

January 17th, 2024

Overview

- ▶ Motivation
- ▶ Original C'est
- ▶ `constexpr` Metamath
- ▶ C'est 2
- ▶ Hands on with C'est 2

Motivation

- ▶ C++ Template Metaprogramming: powerful and expressive
- ▶ Numerous substantial projects are implemented within this idiom
- ▶ But its syntax is not that of standard C++ runtime code
 - ▶ ...while the syntax of the `constexpr` idiom *is*
- ▶ `constexpr` code with UB is likely to produce a compile error
 - ...perhaps useful for cyber security; with no external tools
- ▶ As more features become `constexpr`-friendly, opportunities arise to:
 1. Repurpose decades of existing C++ runtime programs;
 2. Utilise the knowledge of traditional C++ runtime developers

A constexpr schedule?

- ▶ New **constexpr** language features appear in each C++ release
...perhaps in time it will all be *constexpr*?
- ▶ For reference, language features *unavailable* at compile-time include
 - ▶ Virtual inheritance; mutable global variables; **goto** statements
- ▶ But, many language features *are* now available (e.g. in C++26)
- ▶ The C++ standard is updated around every 3 years
- ▶ Typically, a library unit *can* become *constexpr* in the next release
e.g. Proposal P2231 for C++23's **constexpr** `std::optional` and `std::variant` builds on C++20's adoption of P1330 and P0784
- ▶ But the C++ standard library is huge
 - ▶ Each ISO C++ proposal needs motivation, effort and patience
 - ▶ Many library components could have been made *constexpr* years ago

A `constexpr` schedule?

- ▶ New `constexpr` language features appear in each C++ release
...perhaps in time it will all be *constexpr*?
- ▶ For reference, language features *unavailable* at compile-time include
 - ▶ Virtual inheritance; mutable global variables; `goto` statements
- ▶ But, many language features *are* now available (e.g. in C++26)
- ▶ The C++ standard is updated around every 3 years
- ▶ Typically, a library unit *can* become *constexpr* in the next release
e.g. Proposal P2231 for C++23's `constexpr` `std::optional` and `std::variant` builds on C++20's adoption of P1330 and P0784
- ▶ But the C++ standard library is huge
 - ▶ Each ISO C++ proposal needs motivation, effort and patience
 - ▶ Many library components could have been made *constexpr* years ago

The C++ standard library is the first dependency of most projects

Could a non-standard extension offer additional *constexpr*? ...

C'est (Legacy Version)

- ▶ C'est: a non-production header library of *constexpr* C++ classes
...named after those in the C++ standard library
- ▶ The C++ standard library may in time be entirely *constexpr*
...meanwhile, the C'est library can be used today
- ▶ Incomplete support for: `forward_list`, `list`, `set`, `map`, `queue`, `deque`, `istringstream`, `unique_ptr`, `shared_ptr`, `exception` & `function`
- ▶ C'est is not standalone: `libstdc++` is required; and its code is used
- ▶ Established *constexpr* entities from `libstdc++` are wrapped within the `cest` namespace; e.g. `<algorithm>`, `<numeric>`, `vector`, `string`, `array`, `optional`, `pair`, `variant` ...
(C'est also has its own `vector` and `string`)
- ▶ Supports recent versions of `g++` ($\approx 12.2.0$) and `clang++` ($\approx 14.0.6$)
- ▶ The project's Github repository: <https://github.com/SCT4SP/cest>

A Simple C'est Example

- ▶ IO commands such as `std::cout` output nothing (during CE)

```
constexpr bool doit() {  
    using namespace cest;  
  
    string str = "Hello";  
    vector<int> v{1, 2, 3};  
    deque<int> dq{2, 3, 4};  
    set<int> s;  
  
    set_intersection(dq.begin(), dq.end(), v.begin(), v.end(),  
                    inserter(s, s.end()));  
    function<int()> f =  
        [&] { return accumulate(s.begin(), s.end(), 0); };  
    auto x = f();  
    cout << str << " World " << x << endl;  
  
    return 5 == x;  
}
```

ctcheckmm-cest: A constexpr C++ Metamath Database Verifier

- ▶ Metamath: a small formal language to express maths theorems
 - ▶ ...accompanied by proofs, and tools for their verification
- ▶ Over a dozen proof verifiers are listed at <https://us.metamath.org>
- ▶ checkmm: A C++ verifier by Eric Schmidt
 - ▶ 1400 lines of C++ in one source file: `checkmm.cpp`
 - ▶ Makes extensive use of the C++ standard library; 14 headers
 - ▶ Containers: `queue`, `string`, `set`, `deque`, `vector`, `pair`, `map`
 - ▶ The C++ std algorithm library's `set_intersection` and `find`
 - ▶ IO operations involving `std::cout` and `std::cerr`
 - ▶ ...and assorted standalone functions
- ▶ ctcheckmm: a constexpr version:
<https://github.com/pkeir/ctcheckmm>

Either CT & istream or RT & ifstream

```
#define xstr(s) str(s)
#define str(s) #s

constexpr int ce_app_run()
{
    checkmm app;

#ifdef MMFILEPATH
    cest::string txt =
#include xstr(MMFILEPATH)
;
    // Non-default arg 2 here:
    int ret = app.run("", txt);
#else
    int ret = 0;
#endif

    return ret;
}
```

```
int main(int argc, char ** argv)
{
    if (argc != 2)
    {
        cest::cerr
            << "Syntax: checkmm <filename>"
            << cest::endl;
        return -1;
    }

    static_assert(0 == ce_app_run());

    checkmm app;
    int ret = app.run(argv[1]);

    return ret;
}
```

Changes Applied to allow Constant Evaluation (C'est)

- ▶ Added the `constexpr` qualifier to all functions
- ▶ Changed global vars to class members of a simple struct (`checkmm`)
- ▶ Free functions are changed to members of `checkmm`
- ▶ One static function-scope var is also changed to a class member
- ▶ Compile-time file input works by `std::string init: readtokens` now accepts a second `std::string` parameter: used if it isn't empty
- ▶ File-includes within mm database files are not supported when processing at compile-time; an exception is thrown if this occurs
- ▶ C'est file includes: e.g. `#include "cest/vector.hpp"` rather than `#include <vector>`
- ▶ A script places C++11-style raw string literal delimiters before & after mm file contents. A preprocessor macro `MMFILEPATH` is then set to the script's output, during C++ compilation (e.g. `-DMMFILEPATH=peano.mm.raw`)

C'est 2

- ▶ Imagine porting a large, legacy C++ runtime library
- ▶ Do we really want to change the `std` namespace everywhere? *
- ▶ **C'est 2** is a fork of GNU libstdc++ (GCC) on Github
- ▶ Supports all of the classes supported by the legacy version of C'est
- ▶ Repository URL is <https://github.com/SCT4SP/gcc>
- ▶ `basic_ios` inheritance changed; for some `constexpr` `istream` **
 - ▶ Due to the lack of `constexpr` support for virtual inheritance
 - ▶ So, need to build the compiler for programs which also run at runtime
- ▶ master branch pulls from upstream GCC
 - C'est 2 uses the modified `constexpr-std-headers` branch
- ▶ The focus to date has been on supporting the Metamath verifier
- ▶ Also led to prototyping of ISO C++ proposal P3037 targeting C++26

* n.b. We do still need to add the `constexpr` keyword ... to the functions of the library we target for porting.

** Changed inheritance of `basic_ios` from `basic_ostream` to non-virtual, and altered `basic_istream` to inherit from `basic_ostream` rather than virtually from `basic_ios` (commit 6e2f751).

- ▶ Now again running at compile-time, with C'est 2
- ▶ Within the same repo: <https://github.com/pkeir/ctcheckmm>
- ▶ Did much change in the implementation? ...

Changes Applied to allow Constant Evaluation (C'est 2)

- ▶ Added the `constexpr` qualifier to all functions
- ▶ Changed global vars to class members of a simple struct (`checkmm`)
- ▶ Free functions are changed to members of `checkmm`
- ▶ One static function-scope var is also changed to a class member
- ▶ Compile-time file input works by `std::string init: readtokens` now accepts a second `std::string` parameter: used if it isn't empty
- ▶ File-includes within mm database files are not supported when processing at compile-time; an exception is thrown if this occurs
- ▶ C'est file includes: e.g. ~~`##include "cest/vector.hpp"`~~ rather than ~~`##include <vector>`~~
- ▶ A script places C++11-style raw string literal delimiters before & after mm file contents. A preprocessor macro `MMFILEPATH` is then set to the script's output, during C++ compilation (e.g. `-DMMFILEPATH=peano.mm.raw`)

Try C'est 2

To obtain a *current* C++26 **compiler**, choose one:

1. Build from source at <https://github.com/SCT4SP/gcc>
2. On 64-bit Ubuntu, download the Github binary release [here](#)
3. For non-runtime code execution, obtain Debian GCC package [here](#)

```
wget --content-disposition http://kayari.org/gcc-latest/gcc-latest.deb  
sudo dpkg -i gcc-latest_XXXXXXXXXXXXX.deb  
dpkg -L gcc-latest
```

Then either:

```
export LD_LIBRARY_PATH=/opt/gcc-latest/lib64:$LD_LIBRARY_PATH  
/opt/gcc-latest/bin/g++
```

or just...

```
/opt/gcc-latest/bin/g++ -Wl,-rpath,"/opt/gcc-latest/lib64:$LD_LIBRARY_PATH"
```

Then set environment variable CXX26_ROOT to the root. For example:

```
export CXX26_ROOT=/opt/gcc-latest # n.b. Contains `bin` and `lib64` dirs
```

Try C'est 2

For the **Header Files**:

1. Unzip **cest2-headers-v0.0.1.zip** from the Github release [here](#)
2. Assign an environment variable `CEST2_INCLUDE` to its subdirectory:
 - ▶ `constexpr-std-headers/include`

For example:

```
export CEST2_INCLUDE = /my/dir/constexpr-std-headers/include
```

Try C'est 2

A common compiler invocation (using the provided everything.cpp):

```
$CXX26_ROOT/bin/g++ -std=c++26 -Winvalid-constexpr  
-Wl,-rpath,"$CXX26_ROOT/lib64:$LD_LIBRARY_PATH" -I $CEST2_INCLUDE/c++/14.0.0  
-I $CEST2_INCLUDE/c++/14.0.0/x86_64-pc-linux-gnu -L $CXX26_ROOT/lib64  
-D_GLIBCXX_CEST_CONSTEXPR=constexpr -D_GLIBCXX_CEST_VERSION=1  
-fsanitize=address -static-libasan -fconstexpr-ops-limit=2147483647  
-fconstexpr-loop-limit=2147483647 everything.cpp
```


Try C'est 2

Via the ctcheckmm repo, prepare the peano Metamath database:

```
https://github.com/pkeir/ctcheckmm.git
wget https://raw.githubusercontent.com/metamath/set.mm/develop/peano.mm
bash delimit.sh peano.mm
```

Then, verify peano.mm, by building **ctcheck-std.cpp**
(by setting the MMFILEPATH macro):

```
$CXX26_ROOT/bin/g++ -std=c++26 -Winvalid-constexpr
-Wl,-rpath,"$CXX26_ROOT/lib64:$LD_LIBRARY_PATH" -I $CEST2_INCLUDE/c++/14.0.0
-I $CEST2_INCLUDE/c++/14.0.0/x86_64-pc-linux-gnu -L $CXX26_ROOT/lib64
-D_GLIBCXX_CEST_CONSTEXPR=constexpr -D_GLIBCXX_CEST_VERSION=1
-fsanitize=address -static-libasan -fconstexpr-ops-limit=2147483647
-fconstexpr-loop-limit=2147483647 -DMMFILEPATH=peano.mm.raw
ctcheckmm-std.cpp
```

- ▶ Try some other *.mm databases from the Metamath website
- ▶ Add some syntax errors

Acknowledgements

UWS UNIVERSITY OF THE
WEST *of* SCOTLAND

RSE *The Royal Society
of Edinburgh*
KNOWLEDGE MADE USEFUL