`constexpr` or the evolution of const-ness in recent years

Adam Graliński

C++ FFFE, October 2021

A quick reminder

You are welcome to:

- interrupt me
- ask questions immediately :-)

cv-type qualifiers

en.cppreference.com/w/cpp/language/cv

cv-type qualifiers

en.cppreference.com/w/cpp/language/cv

- **`const`**
  - defines that the type is *constant*

```
const auto x = 13;
```

cv-type qualifiers

en.cppreference.com/w/cpp/language/cv

- **`const`**
  - defines that the type is *constant*

```
const auto x = 13;
```

- **`(no qualifier)`**
  - the type is* a standard *variable*

```
auto x = 13;
```

cv-type qualifiers

en.cppreference.com/w/cpp/language/cv

- `const`
  - defines that the type is *constant*

```
const auto x = 13;
```

- `(no qualifier)`
  - the type is* a standard *variable*

```
auto x = 13;
```

- `volatile`
  - the type is *volatile*

```
volatile auto x = 13;
```

cv-type qualifiers

en.cppreference.com/w/cpp/language/cv

- **`const`**
  - defines that the type is *constant* ← (non-mutable)

```
const auto x = 13;
```

- **`(no qualifier)`**
  - the type is* a standard *variable* ← (mutable)

```
auto x = 13;
```

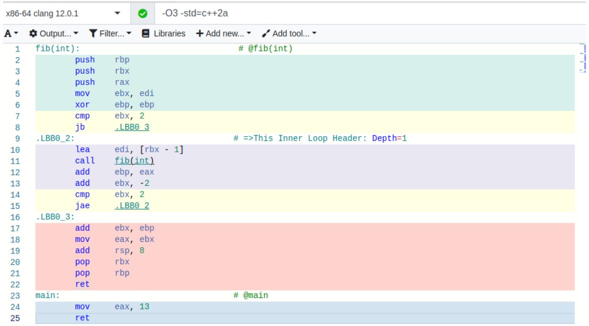
- **`volatile`**
  - the type is *volatile* ← (extremely mutable)

```
volatile auto x = 13;
```

cv-type qualifiers demo

```
auto fib(int n) {
  if (n == 0) {
    return 0;
  }
  if (n == 1) {
    return 1;
  }
  return fib(n-1) + fib(n-2);
}

int main()
{
  int rv = 13;
  return rv;
}
```



```
fib(int):                               # @fib(int)
        push    rbp
        push    rbx
        push    rax
        mov     ebx, edi
        xor     ebp, ebp
        cmp     ebx, 2
        jb      .LBB0_3
.LBB0_2:                                # =>This Inner Loop Header: Depth=1
        lea     edi, [rbx - 1]
        call    fib(int)
        add     ebp, eax
        add     ebx, -2
        cmp     ebx, 2
        jae     .LBB0_2
.LBB0_3:
        add     ebx, ebp
        mov     eax, ebx
        add     rsp, 8
        pop     rbx
        pop     rbp
        ret
main:                                   # @main
        mov     eax, 13
        ret
```

`constexpr` — C++11

en.cppreference.com/w/cpp/language/constexpr

`constexpr` — specifies that the value of a variable or function can appear in constant expressions

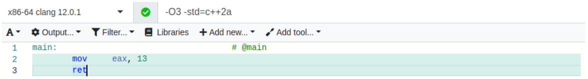
A **constexpr function** must satisfy the following requirements:

- it must not be virtual (until C++20)
- its return type (if any) must be a LiteralType
- each of its parameters (if any) must be a LiteralType
- for constructor (...), the class must have no virtual base classes

switching to a constexpr...

```
constexpr auto fib(int n) {
  if (n == 0) {
    return 0;
  }
  if (n == 1) {
    return 1;
  }
  return fib(n-1) + fib(n-2);
}

int main()
{
  constexpr int rv = fib(7);
  return rv;
}
```



```
main:                                   # @main
        mov     eax, 13
        ret
```

constexpr or consteval?

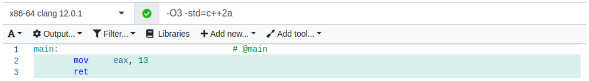
godbolt.org/z/4cv71W

constexpr or consteval?

godbolt.org/z/4cv71W

```
consteval auto fib(int n) {
  if (n == 0) {
    return 0;
  }
  if (n == 1) {
    return 1;
  }
  return fib(n-1) + fib(n-2);
}

int main()
{
  return fib(7);
}
```



x86-64 clang 12.0.1 -O3 -std=c++2a

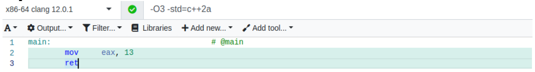
```
main:                                   # @main
        mov     eax, 13
        ret
```

Caveat: exceptions

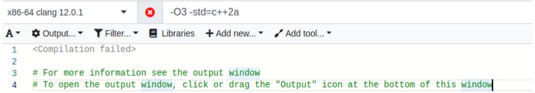
the function body must not contain: a try-block* relaxed for C++20

```
constexpr auto fib(int n) {
  if (n<0) {
    throw "Must␣use␣nonnegative␣integers";
  }
  if (n == 0) {
    return 0;
  }
  if (n == 1) {
    return 1;
  }
  return fib(n-1) + fib(n-2);
}

int main()
{
  constexpr int rv = fib(7);
  return rv;
}
```


x86-64 clang 12.0.1 -O3 -std=c++2a

```
main:                                   # @main
        mov     eax, 13
        ret
```


x86-64 clang 12.0.1 -O3 -std=c++2a

```
<Compilation failed>

# For more information see the output window
# To open the output window, click or drag the "Output" icon at the bottom of this window
```

Caveat: stepping outside LiteralTypes

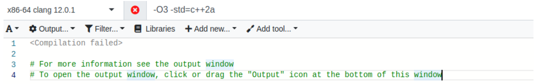
the function's signature...

- ...return type must be a LiteralType
- ...each of its parameters must be a LiteralType

```
struct Point {
  int x, y;
  Point(int x = 0, int y = 0): x(x), y(y) {}
};

int main()
{
  constexpr auto myDouble = 0.13; // OK

  constexpr auto myPoint = Point{0, 13};
  // wrong, Point is not a LiteralType
}
```



```
<Compilation failed>

# For more information see the output window
# To open the output window, click or drag the "Output" icon at the bottom of this window
```

```
error: constexpr variable cannot have non-literal type 'const Point'
```

Caveat: cryptic arcane stuff from the „do not open” bag

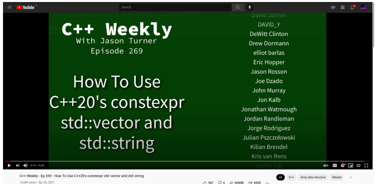
the function body must not contain:

- `goto` statements
- labels other than `case` and `default`
- asm blocks

But you're already **not using them**.

Taste of the future: `constexpr std::string`, `std::array`, `std::vector`

`C++ Weekly` #269: youtube.com/watch?v=cuFILbHp-RA



C++ Weekly
With Jason Turner
Episode 269

How To Use
C++20's constexpr
std::vector and
std::string

David Steffen
DAVID_Y
DeWitt Clinton
Drew Dormann
elliot barlas
Eric Hopper
Jason Rosson
Joe Dzado
John Murray
Jon Kalb
Jonathan Watmough
Jordan Randleman
Jorge Rodriguez
Julian Pszczołowski
Kilian Brendel
Kris van Rens

C++ Weekly - Ep 269 - How To Use C++20's constexpr std::vector and std::string

- Sort your `std::vector` of `std::strings` at compile time!
- `std::accumulate()` your `std::vector` of doubles at compile time!
- -std=C++20 and not yet implemented by your compiler vendor!
  - ...but some fastring insider previev of `MSVC` has/had it!
  - ...while both `clang++` & `g++` support constexpr constructors now

Taste of the future: `consteval if` (C++23)

en.cppreference.com/w/cpp/language/if#Consteval_if

Key takeaways

- Make your constant expressions **`const`** or **`constexpr`**
- Make your functions **`consteval`** or **`constexpr`** where possible
- No downsides!
- Be aware of the ,,slow march of progress” across C++ standards
- Be aware of the lag in compiler implementations for C++20 features

Thank you!