zühlke
empowering ideas

HAM!

Pillars of Modern C++

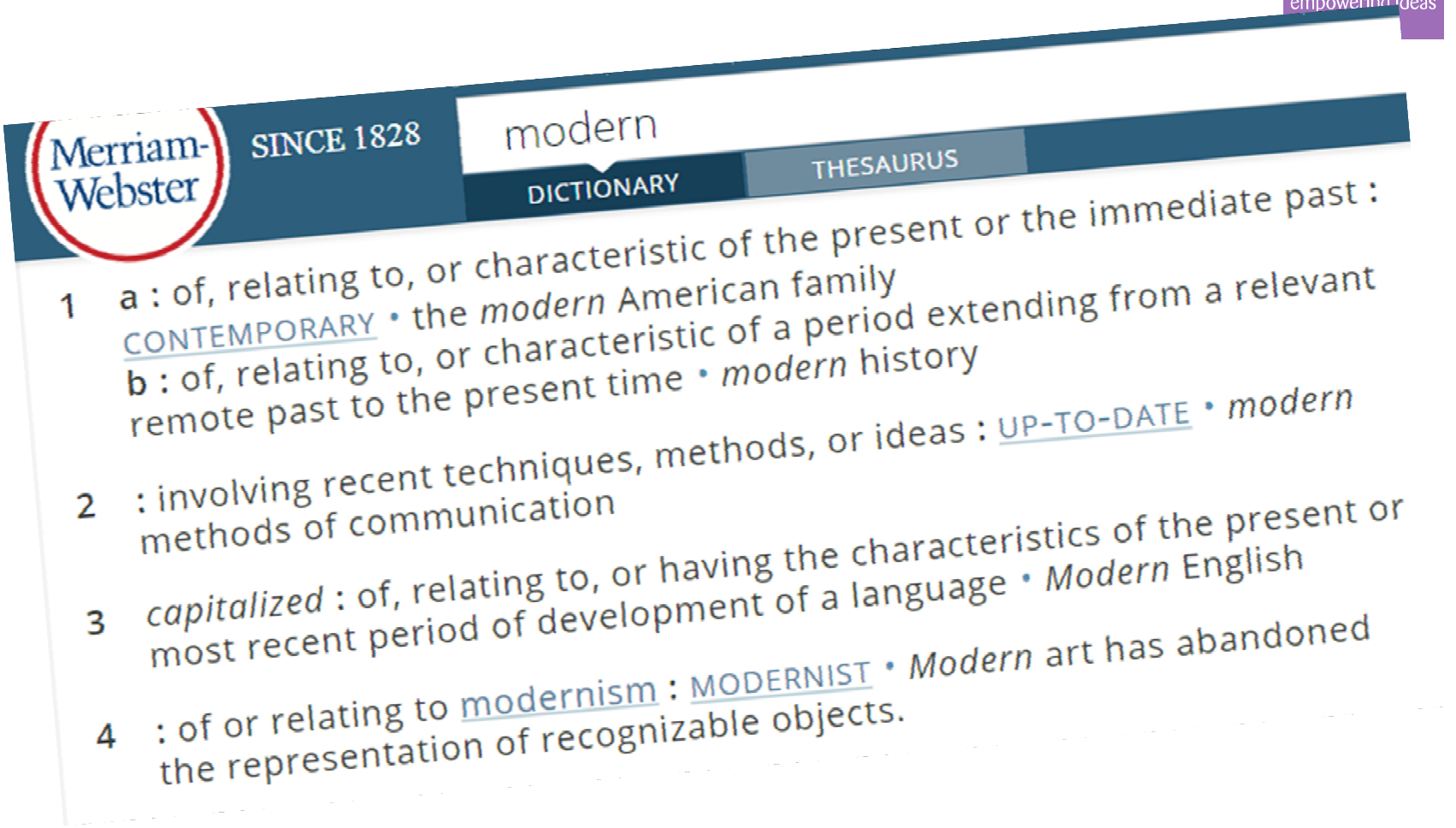
@arne_mertz

History

- 1979 “C with classes” (influenced by Simula, ALGOL68,…)
- 1983 renamed to “C++”, Cfront compiler
- 1985 “The C++ Programming Language, 1st Ed.”
- 1990 C++ 2.0 & “The Annotated C++ Reference Manual”
- 1998 *ISO/IEC 14882:1998*
- 2003 *ISO/IEC 14882:2003* „Bugfix release“
- 2011 *ISO/IEC 14882:2011* „feels like a new language“
- 2014 *ISO/IEC 14882:2014* „minor features“
- 2017 *ISO/IEC 14882:2017* „major features“


Photo by Julia Kryuchkova CC BY-SA 2.5

Modern C++?



Merriam-Webster SINCE 1828

modern

DICTIONARY THESAURUS

1 a : of, relating to, or characteristic of the present or the immediate past : CONTEMPORARY • the *modern* American family
b : of, relating to, or characteristic of a period extending from a relevant remote past to the present time • *modern* history

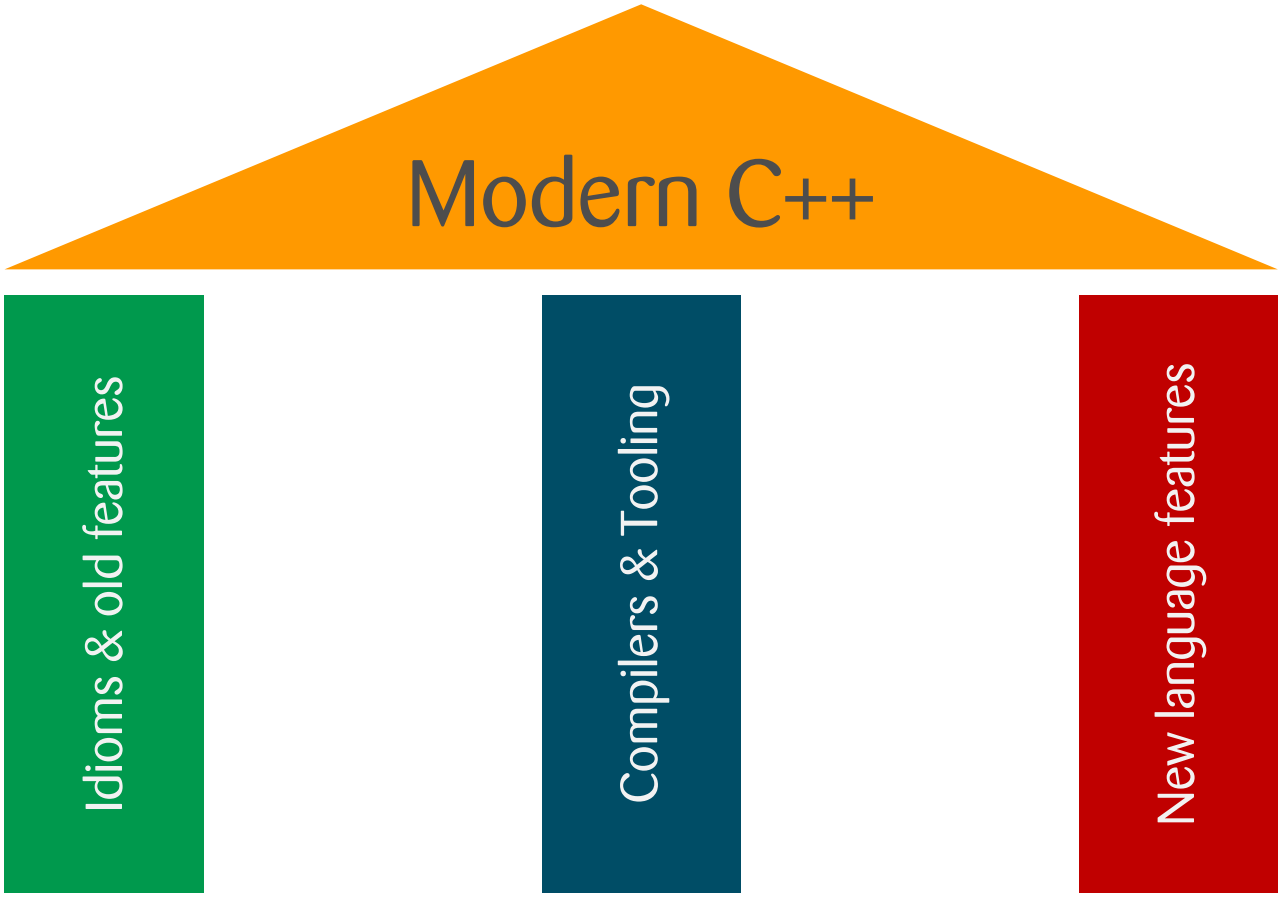
2 : involving recent techniques, methods, or ideas : UP-TO-DATE • *modern* methods of communication

3 *capitalized* : of, relating to, or having the characteristics of the present or most recent period of development of a language • *Modern* English

4 : of or relating to modernism : MODERNIST • *Modern* art has abandoned the representation of recognizable objects.

“Modern C++” != “New Standards”

zühlke
empowering ideas



RAII

Resource Acquisition Is Initialization

One of the key C++ language features:

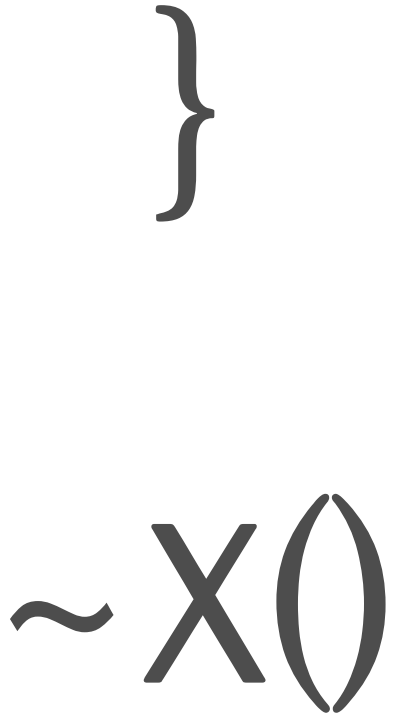
Deterministic Object Lifetime

```
}
~X()
```

RAII

Resource Acquisition Is Initialization

- Use destructors to clean up resources
  - Memory
  - File handles
  - Mutex locks
  - Database connections
  - Network sockets
  - …



RAII

~~Resource~~ Responsibility Acquisition Is Initialization

- … but also non-resources
  - Close XML tags
  - Set the wait cursor back to normal cursor
  - Reactivate interrupts

  }

  - … or (un)do anything else that needs to be (un)done reliably in the end

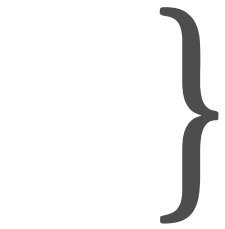
  ~X()

  - This is why C++ does not have or need **`finally`**

RAII

Resource Acquisition Is Initialization

```
class ScopePrinter {
public:
    std::string message;
    ScopePrinter(std::string const& msg) : message(msg) {
        std::cout << "START " << message << "\n";
    }

    ~ScopePrinter() { std::cout << "END " << message << "\n"; }
};

int main() {
    ScopePrinter sp1("main");       // START main
    {
        ScopePrinter sp2("inner"); // START inner
    }                               // END inner
    ScopePrinter sp3("main2");      // START main2
}                                   // END main2
                                    // END main
```



RAII

Resource Acquisition Is Initialization

- In the standard library:
- std::vector (memory - for dynamic arrays) C++98
- std::fstream (file I/O) C++98
- std::unique_ptr (memory, unique owner) C++11
- std::shared_ptr (memory, shared owner, ref-counting) C++11
- std::lock_guard, std::unique_lock, … (mutexes) C++11
- … and more

RAII

Resource Acquisition Is Initialization

```
class LinkedList {
    struct Node {
        int value;
        std::unique_ptr<Node> next;
        Node(int v, std::unique_ptr<Node> n)
            : value{v}, next{std::move(n)}
        {}
    };

    std::unique_ptr<Node> head;
public:
    void push(int i) {
        head = std::make_unique<Node>(i, std::move(head));
    }
    void pop() {
        head = std::move(head->next);
    }
};
```

}

~X()

Strong Typing

Is C++ type-safe?


```
void sendMessage(int messageID, int receiverID, int priority);

sendMessage(SEVERE, MSG_SOME_ERROR, myReceiver); //oops
```

Strong Typing

… as type-safe as we make it!


```
enum Priority {
    SEVERE,
    //...
};

struct MessageID {
    int id;
};

struct ReceiverID {
    int id;
};

void sendMessage(MessageID messageID, ReceiverID receiverID, Priority priority);

sendMessage(SEVERE, MSG_SOME_ERROR, myReceiver);

//error: could not convert 'SEVERE' from 'Priority' to 'MessageID'
//error: could not convert 'MSG_SOME_ERROR' from 'const MessageID' to 'ReceiverID'
//error: cannot convert 'ReceiverID' to 'Priority'
```

What does it cost?

Because performance matters!

Nothing.

(at runtime)

Zero-Cost Abstractions

- Language/library design philosophy: “Zero overhead principle”
  - Don’t pay for what you don’t use
  - Can’t reasonably be written more efficiently by hand

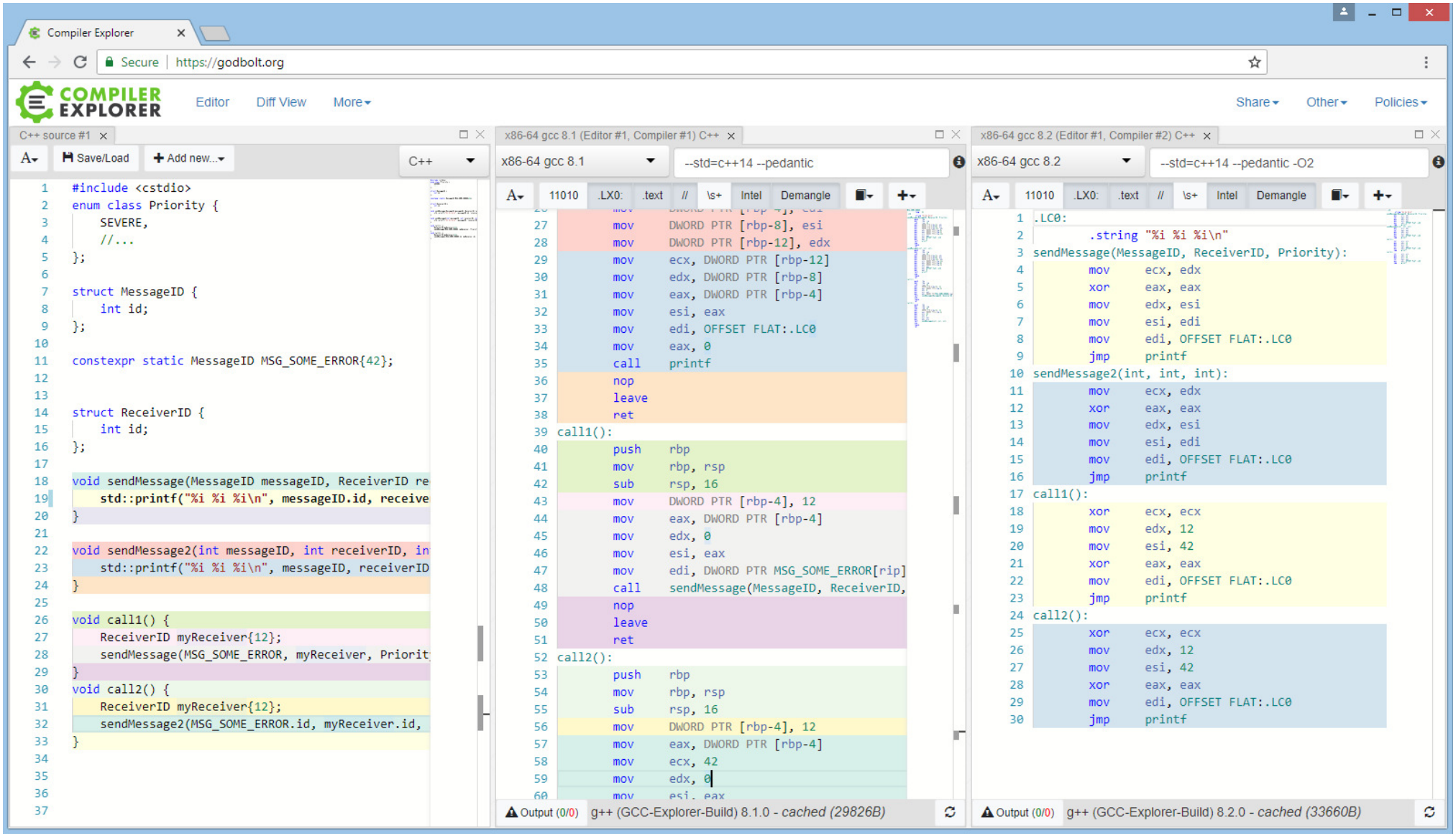
- Modern C++ compilers have very good optimizers
  - Inlining of functions
  - Elimination of variables
  - Even at link time

Compilers

- Trust the compiler, not the myths!
- Measure before you manually “optimize”

And after!

- Maybe have a look at the assembler: godbolt.org


Compiler Explorer
Secure | https://godbolt.org
COMPILER EXPLORER
Editor
Diff View
More
Share
Other
Policies
C++ source #1
Save/Load
Add new...
C++
#include <cstdio>
enum class Priority {
SEVERE,
//...
};
struct MessageID {
int id;
};
constexpr static MessageID MSG_SOME_ERROR{42};
struct ReceiverID {
int id;
};
void sendMessage(MessageID messageID, ReceiverID re
std::printf("%i %i %i\n", messageID.id, receive
}
void sendMessage2(int messageID, int receiverID, in
std::printf("%i %i %i\n", messageID, receiverID
}
void call1() {
ReceiverID myReceiver{12};
sendMessage(MSG_SOME_ERROR, myReceiver, Priorit
}
void call2() {
ReceiverID myReceiver{12};
sendMessage2(MSG_SOME_ERROR.id, myReceiver.id,
}
x86-64 gcc 8.1 (Editor #1, Compiler #1) C++
x86-64 gcc 8.1
--std=c++14 --pedantic
11010
.LX0:
.text
//
\s+
Intel
Demangle
mov DWORD PTR [rbp-8], esi
mov DWORD PTR [rbp-12], edx
mov ecx, DWORD PTR [rbp-12]
mov edx, DWORD PTR [rbp-8]
mov eax, DWORD PTR [rbp-4]
mov esi, eax
mov edi, OFFSET FLAT:.LC0
mov eax, 0
call printf
nop
leave
ret
call1():
push rbp
mov rbp, rsp
sub rsp, 16
mov DWORD PTR [rbp-4], 12
mov eax, DWORD PTR [rbp-4]
mov edx, 0
mov esi, eax
mov edi, DWORD PTR MSG_SOME_ERROR[rip]
call sendMessage(MessageID, ReceiverID,
nop
leave
ret
call2():
push rbp
mov rbp, rsp
sub rsp, 16
mov DWORD PTR [rbp-4], 12
mov eax, DWORD PTR [rbp-4]
mov ecx, 42
mov edx, 0
Output (0/0)
g++ (GCC-Explorer-Build) 8.1.0 - cached (29826B)
x86-64 gcc 8.2 (Editor #1, Compiler #2) C++
x86-64 gcc 8.2
--std=c++14 --pedantic -O2
.LC0:
.string "%i %i %i\n"
sendMessage(MessageID, ReceiverID, Priority):
mov ecx, edx
xor eax, eax
mov edx, esi
mov esi, edi
mov edi, OFFSET FLAT:.LC0
jmp printf
sendMessage2(int, int, int):
mov ecx, edx
xor eax, eax
mov edx, esi
mov esi, edi
mov edi, OFFSET FLAT:.LC0
jmp printf
call1():
xor ecx, ecx
mov edx, 12
mov esi, 42
xor eax, eax
mov edi, OFFSET FLAT:.LC0
jmp printf
call2():
xor ecx, ecx
mov edx, 12
mov esi, 42
xor eax, eax
mov edi, OFFSET FLAT:.LC0
jmp printf
Output (0/0)
g++ (GCC-Explorer-Build) 8.2.0 - cached (33660B)

Is C++ an Object Oriented Language?

- It's procedural. (But not "C with classes")
- It's *also* object oriented. (But not "Java without the GC")
- It's *also* functional. (But not "Haskell with poor syntax")
- It's also generic.

C++ is called a *multi-paradigm language*

Pick the aspect that best solves the problem at hand.

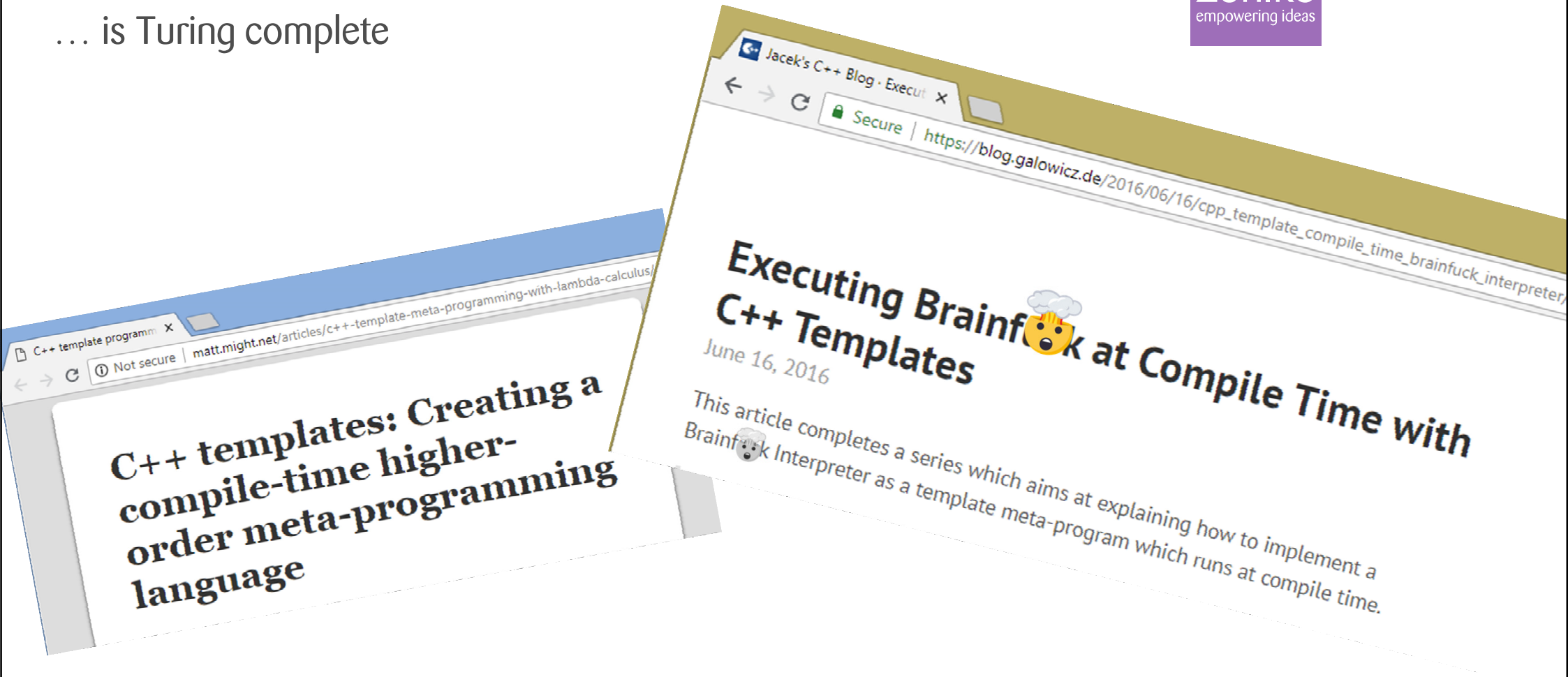
Speaking of Generic…

- In C++03: *Template Metaprogramming* is a powerful technique for compile time logic
- *Metafunctions* map types and values to other types and/or values

```
template <unsigned int N>
struct fib {
    const static unsigned int value = fib<N-1>::value + fib<N-2>::value;
};

template <> struct fib<0> {
    const static unsigned int value = 1;
};
template <> struct fib<1> {
    const static unsigned int value = 1;
};
```

Template Metaprogramming

… is Turing complete



emojis by EmojiOne

Modern language features

C++11, C++14, C++17, …

- Language and library
- “Quality of life”
- Standardizing existing extensions
- Access to compiler intrinsics

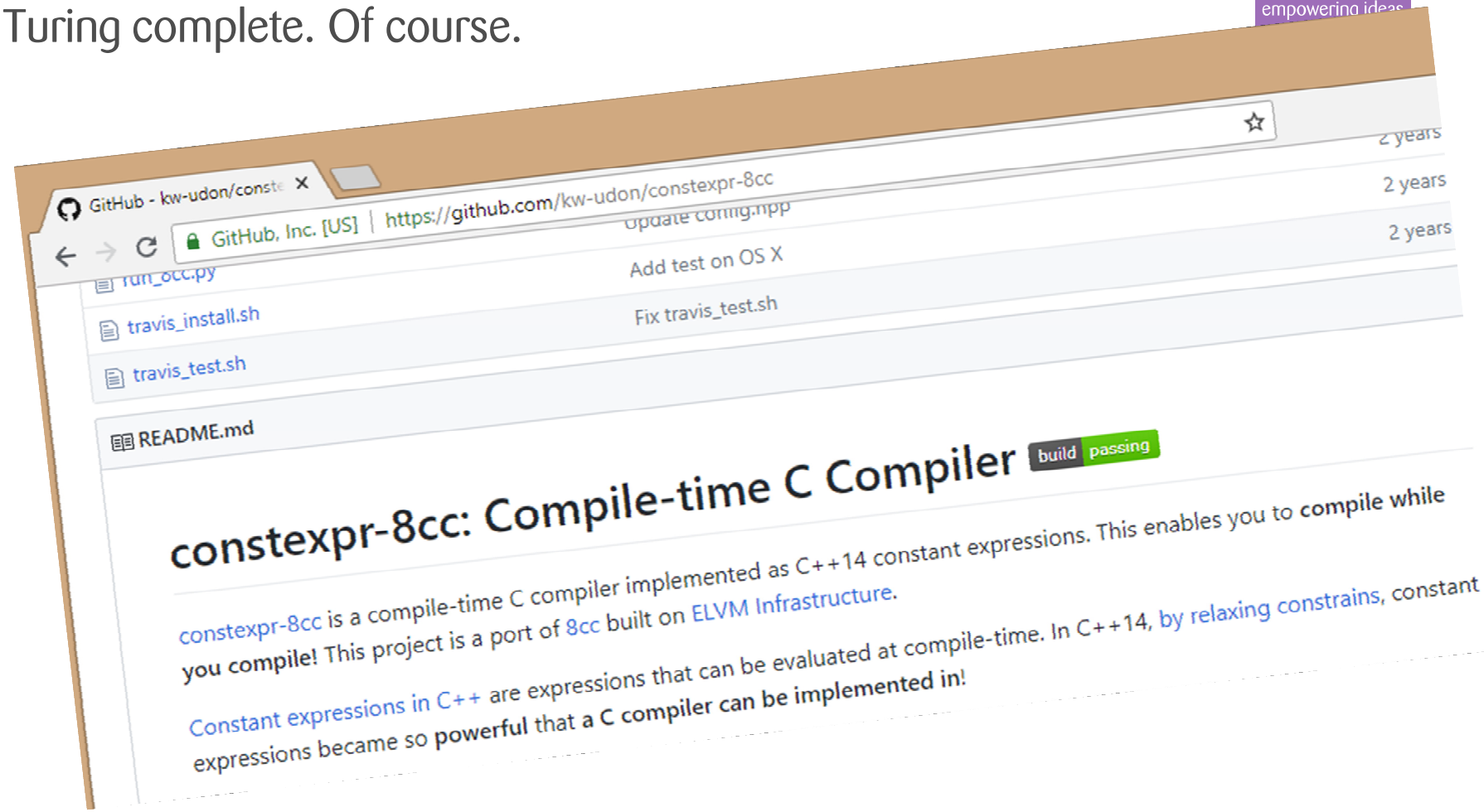
More compile time logic

- Since C++11/14: keyword `constexpr`
- "Normal" C++ code that can be executed at compile time
  - Can not allocate memory, throw exceptions, etc.
  - Classes can have constexpr constructors etc.

```
constexpr int fib(int n) {
    if (n < 0) throw std::logic_error("Negative argument!");
    if (n < 2) return 1;
    return fib(n-1) + fib(n-2);
}
```

Constexpr

… is Turing complete. Of course.



More templates

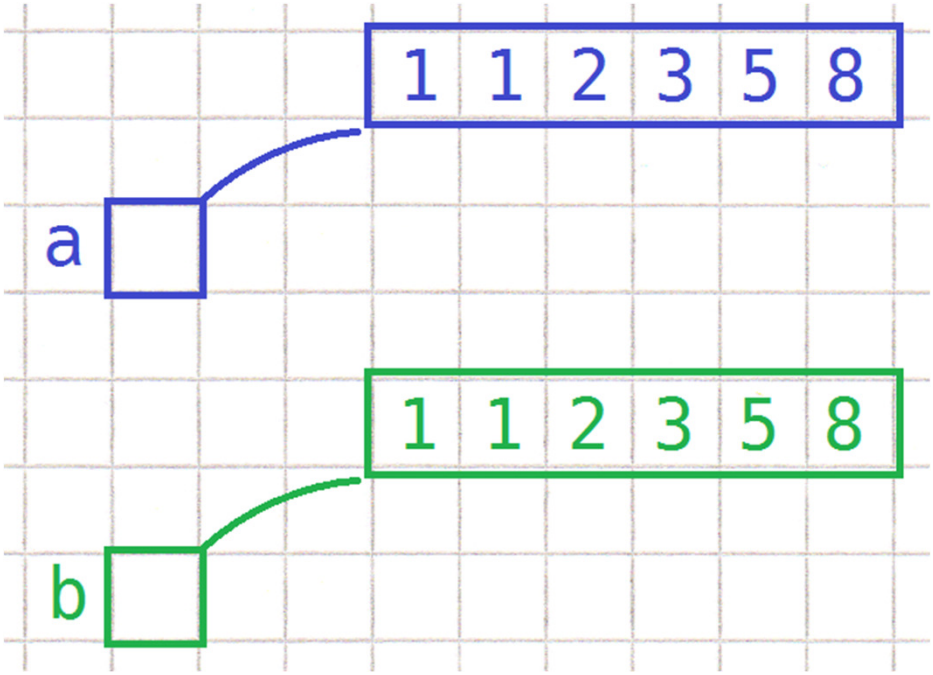
```
template <class T>
void print(T const& t) {
    std::cout << t << '\n';
}

template <class Head, class... Tail>
void print(Head const& head, Tail const&... tail) {
    print(head);
    print(tail...);
}

int main() {
    print("Hello, ", 42, 2.71828);
    //print<char const*, int, double>
}
```

```
template <class T>
const static std::size_t BYTE_COUNT
    = sizeof(T);


template <class T>
using ByteArray
    = std::array<uint8_t, BYTE_COUNT<T>>;
```

Move semantics

Before (C++03):

```
std::vector<int> create();

class C {
    std::vector<int> m_data;
public:
    C(std::vector<int> const& data)
        : m_data(data)
    {}
};

int main() {
    auto data = create();
    C c(data);
}
```



Move semantics

C++11 adds rvalue references

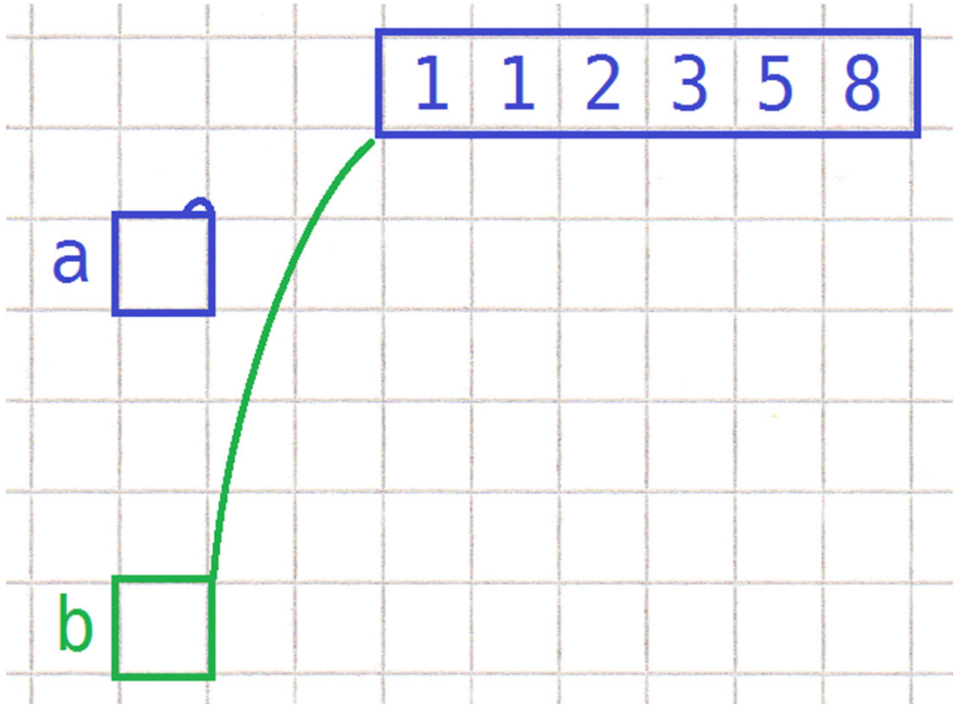
- Rvalue: r for “right side of assignment”
- Rvalue references bind to temporaries
  - E.g. function return values
  - `void f(int&& i)`
- `std::move`: “cast to rvalue”

```
class X;

void f(X && rvalue);
void f(X const& lvalue);

X g();

int main() {
  X x;
  f(x);   //lvalue
  f(g()); //rvalue
  f(X()); //rvalue
  f(std::move(x)); //rvalue
}
```

Move semantics

Copy and Move Constructor

```
class vector {
  int* data;
public:
  vector(vector const& other)
    : data(deepcopy(other))
  {}

  vector(vector && other)
    : data(nullptr)
  {
    swap(data, other.data);
  }
};
```

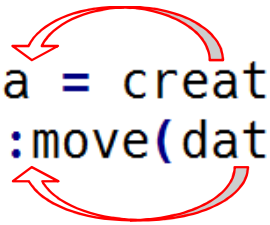


Move semantics

Then vs. Now

```
std::vector<int> create();

class C {
    std::vector<int> m_data;
public:
    C(std::vector<int> const& data)
        : m_data(data)
    {}
};

int main() {
    auto data = create();
    C c(data);
}
```

```
std::vector<int> create();

class C {
    std::vector<int> m_data;
public:
    C(std::vector<int> && data)
        : m_data(std::move(data))
    {}
};

int main() {
    auto data = create();
    C c(std::move(data));
}
```



Quality of life features

Auto type deduction (C++11)

```
auto i = myVector.begin();
```

Trailing return types (C++11)

```
template <class T, class U>
auto sum(T const& t, U const& u) -> decltype(t+u)
{
    return t+u;
}
```

Return type deduction (C++14)

Range-based for loops (C++11)

```
for (auto const& element : myContainer)
{
    std::cout << element << '\n';
}
```

Defaulted and deleted functions (C++11)

```
struct X {
  X(X const& x) = delete;
  X(X && x)     = default;
};
```

Quality of life features


```
std::pair<int, double> f();
auto [i, d] = f();

void g(X* xPtr);
g(0);       //C++03
g(nullptr); //C++11

struct S : Base {
    using Base::Base;

    S(int i) { std::cout << i; };
    S(int i, double d) : S(i) { std::cout << ", " << d; }
};
```

Structured Bindings (C++17)

nullptr (C++11)

Inheriting constructors (C++11)

Delegating constructors (C++11)

Quality of life features

Let's print contents of a map: `(key -> value)`

C++03

```
template <class Map>
void printMap (Map const& m) {
    for (typename Map::const_iterator it = m.begin(); it != m.end(); ++it) {
        typename Map::key_type const& key = it->first;
        typename Map::mapped_type const& value = it->second;
        std::cout << "(" << key << " -> " << value << ")\n";
    }
}
```

C++17

```
template <class Map>
void printMap2 (Map const& m) {
    for (auto const& [key, value] : m) {
        std::cout << "(" << key << " -> " << value << ")\n";
    }
}
```

Lambdas

C++11/14

```
std::copy_if(
  std::begin(oldSeq), std::end(oldSeq),
  std::begin(newSeq),
  [](int i) -> int { return i >= 5; }
);
```

≈

```
class %$@#& {
    auto operator()(int i) -> int
    { return i >= 5; }
};
```

```
int lim = 5;
std::copy_if(
  std::begin(oldSeq), std::end(oldSeq),
  std::begin(newSeq),
  [lim](auto i) { return i >= lim; }
);
```

≈

```
class @%$&# {
    int lim;
    template <class T>
    auto operator()(T i) const
    { return i >= lim; }
};
```

Lambdas

Higher order functions

```
auto filter(int lim) {
    return [lim](int i) {
        return i >= lim;
    };
}


template <class F, class G>
auto compose(F f, G g) {
    return [f, g](auto x){
        return f(g(x));
    }
}
```

And much more…

C++11 feature	Paper(s)	Version	GCC	Clang	MSVC	EDG eccp	Intel C++	IBM XLC++	SunOracle C++	Embarcadero C++ Builder	Qnx	PortlandGroup (PGI)	HP aCC	Digital Mars C++	D
alignas	N2341	c++11	4.8	3.0	19.0*	4.8	15.0	13.1.2*	5.13	Yes	8.6	2015			
alignof	N2341	c++11	4.8	2.9	19.0*	4.8	15.0	13.1.2*	5.13	Yes	8.4	2015			
Atomic operations	N2427	c++11	4.4	3.1	17.0*	Yes	15.0	13.1.2*	5.14	Yes	8.4	2015			
auto	N1984 (v1.0)	c++11	4.4(v1.0)	Yes	16.0*	4.1(v0.9)	11.0(v0.9) 12.0(v1.0)	11.1(v1.0)	5.13	Yes	8.4	2015	A.06.25		
C99 preprocessor	N1653		4.3	Yes	19.0* (partial - buggy variadic macros)	4.1	11.1	10.1	5.5	Yes	8.4	2015	A.06.25	Yes	
constexpr	N2235	c++11	4.6	3.1	19.0* (partial)	4.6	13.0* 14.0	12.1* 13.1	5.13	Yes	8.4	2015	A.06.25		
decltype	v1.0: N2343 v1.1: N3276	c++11	4.3(v1.0) 4.8.1(v1.1)	2.9	16.0*	4.1(v1.0)	11.0(v1.0) 12.0(v1.1)	11.1(v1.0)	5.13	Yes	8.4	2015	A.06.25	8.52(v1.0)	
Defaulted and deleted functions	N2346	c++11	4.4	3.0	18.0*	4.1	12.0	13.1	5.13	Yes	8.4	2015	A.06.25		
Delegating constructors	N1986	c++11	4.7	3.0	18.0*	4.7	14.0	11.1	5.13	Yes	8.4	2015	A.06.25		
Explicit conversion operators	N2437	c++11	4.5	3.0	18.0*	4.4	15.0	12.1	5.13	Yes	8.4	2015	A.06.27		
Extended friend declarations	N1791	c++11	4.7	2.9	16.0* (partial)	4.1	11.1* 12.0	11.1	5.13	Yes	8.4	2015	A.06.25		
extern template	N1987	c++11	3.3	Yes	12.0*	3.9	9.0	11.1	5.13	Yes	8.4	2015	A.06.25		
Forward enum declarations	N2764	c++11	4.6	3.1	17.0*	4.5	11.1* 14.0	12.1	5.13	Yes	8.4	2015			
Inheriting constructors	N2540	c++11	4.8	3.3	19.0*	4.5	15.0	13.1.1*	5.13	Yes	8.4	2015			
Initializer lists	N2672	c++11	4.4	3.1	18.0*	4.5	13.0* 14.0	13.1.2*	5.13	Yes	8.4	2015	A.06.25		
Lambda expressions	v0.9: N2550 v1.0: N2658 v1.1: N2927	c++11	4.5(v1.1)	3.1	16.0*(v1.0) 17.0*(v1.1)	4.1(v1.1)	12.0(v1.1)	13.1.2*	5.13	Yes	8.4	2015	A.06.25		
Local and unnamed types as template parameters	N2657		4.5	2.9	16.0*	4.2	12.0	13.1.2*	5.13	Yes	8.4	2015	A.06.27		
long long	N1811	c++11	Yes	Yes	14.0*	Yes	Yes	Yes	Yes	Yes	8.4	2015	Yes	Yes	
Inline namespaces	N2535	c++11	4.4	2.9	19.0*	4.5	14.0	11.1	5.13	Yes	8.4	2015	A.06.25		
New character types	N2249	c++11	4.4	2.9	19.0*	4.4	12.1* 14.0	13.1.1*	5.13	Yes	8.4	2015	A.06.27	8.52	
Trailing function return types	N2541	c++11	4.4	2.9	16.0*	4.1	12.0	12.1	5.13	Yes	8.4	2015	A.06.27		
nullptr	N2431	c++11	4.6	2.9	16.0*	4.2	12.1	13.1	5.13	Yes	8.4	2015	A.06.27	8.52	
Unicode string literals	N2442	c++11	4.4	3.0	19.0*	4.7	11.0*	10.1* 13.1.1*	5.7	Yes	8.4	2015	A.06.25	8.52	
Raw string literals	N2442	c++11	4.5	Yes	18.0*	4.7	14.0	13.1.1*	5.13	Yes	8.4	2015	A.06.25	8.52	
User-defined literals	N2765	c++11	4.7	3.1	19.0*	4.5	15.0	13.1.2*	5.14	Yes	8.4	2015			
Right angle brackets	N1757	c++11	4.3	Yes	14.0*	4.1	11.0	12.1	5.13	Yes	8.4	2015			
Rvalue references	v1.0: N2118 v2.0: N2844 v2.1: N2844 v3.0: N3053	c++11	4.3(v1.0) 4.5(v2.1) 4.6(v3.0)	Yes	16.0*(v2.0) 17.0*(v2.1) 18.0*(v3.0)	4.3(v3.0)	11.1(v1.0) 12.0(v2.0) 14.0(v3.0)	12.1(v2.1)	5.13	Yes	8.4	2015	A.06.25		
static_assert	N1720	c++11	4.3	2.9	16.0*	4.1	11.0	11.1	5.13	Yes	8.4	2015	A.06.25	8.52	
Strongly-typed enum	N2347	c++11	4.4	2.9	17.0*	4.0	13.0	12.1	5.13	Yes	8.4	2015	A.06.25		
Template aliases	N2258	c++11	4.7	3.0	18.0*	4.2	12.1	13.1.1*	5.13	Yes	8.4	2015	A.06.27		
Thread-local storage	N2659	c++11	4.4* 4.8	2.9* 3.3	16.0* (partial) 18.0*	4.8	11.1* 15.0*	10.1* 13.1.2*	5.9*	Yes	8.4	2015		8.52*	
Unrestricted unions	N2544	c++11	4.6	3.0	19.0*	4.6	14.0*	13.1.2*	5.13	Yes	8.4	2015	A.06.25		
Type traits	N1836	c++11	4.3	3.0	14.0*	4.0	10.0	15.1.3	5.13	Yes	8.4	2015	6.16		
Variadic templates	v0.9: N2242 v1.0: N2555	c++11	4.3(v0.9) 4.4(v1.0)	2.9(v1.0)	18.0*	4.1(v0.9)	12.1(v1.0)	11.1(v0.9)	5.13	Yes	8.4	2015	A.06.27		
Range-for loop	N2930	c++11	4.6	3.0	17.0*	4.5	15.0	13.1.2*	5.13	Yes	8.4	2015	A.06.25		
override and final	v0.8: N2928 v0.9: N3206 v1.0: N3272	c++11	4.7	2.9	14.0* (partial) 17.0*	4.8(v1.0)	12.0(v0.8) 14.0(v1.0)	13.1.1*	5.13	Yes	8.4	2015			
Attributes	N2761	c++11	4.8	3.3	19.0*	4.2	12.1	13.1.1*	5.13	Yes	8.4	2015	A.06.27		
ref-qualifiers	N2439	c++11	4.8.1	2.9	19.0*	4.7	14.0	13.1.3*	5.13	Yes	8.4	2015	A.06.25		
Non-static data member initializers	N2756	c++11	4.7	3.0	18.0*	4.6	14.0	13.1.2*	5.13	Yes	8.4	2015	A.06.25		
Dynamic initialization and destruction with concurrency (magic statics)	N2660	c++11	4.3	2.9	19.0*	Yes	11.1*	13.1.2*	5.13	Yes	8.4	2015	A.06.25		
noexcept	N3050	c++11	4.6	3.0	19.0*	4.5	14.0	13.1.1*	5.13	Yes	8.4	2015	A.06.25		
dynamic pointer safety ISO interface	N2670	c++11				N/A	N/A					N/A			
Money, Time, and Hash/IO manipulators	Time: N2071 Money: N2072	c++11	5	3.6	19.0*	N/A	N/A		5.15			N/A			

C++14 feature	Paper(s)	Version	GCC	Clang	MSVC	EDG eccp	Intel C++	IBM XLC++	SunOracle C++	Embarcadero C++ Builder	Qnx	PortlandGroup (PGI)	HP aCC	Digital Mars C++
Threaded wording for contextual conversions	N3323	c++14-lang	4.9	3.4	18.0*	4.9	16.0	13.1.3*	5.15		8.6	16.1		
Binary literals	N3472	c++14-lang	4.3/4.9	2.9	19.0*	4.10	11.0	13.1.2*	5.14		8.6	2015		
decltype(auto), Return type deduction for normal functions	N3638	c++14-lang	4.8/4.9	3.3/3.4	19.0*	4.9	15.0	13.1.2*	5.15		8.6	16.1		
Initialized/Generalized lambda captures (init-capture)	N3648	c++14-lang	4.5/4.9	3.4	19.0*	4.10	15.0		5.15		8.6	16.1		
Generic (polymorphic) lambda expressions	N3649	c++14-lang	4.9	3.4	19.0*	4.10	16.0	13.1.2*	5.15		8.6	16.1		
Variable templates	N3651	c++14-lang	5	3.4	19.0*	4.11	17.0	13.1.2*	5.15		8.6	17.4		
Extended constexpr	N3652	c++14-lang	5	3.4	19.10*	4.11	17.0	13.1.2*	5.15		8.6	17.4		
Member initializers and aggregates (NSDMI)	N3653	c++14-lang	5	3.3	19.10*	4.9	16.0		5.14		8.6	16.1		
Clarifying memory allocation (avoiding/fusing allocations)	N3664	c++14-lang	N/A	3.4	N/A	N/A			N/A		8.6	17.4		
[[deprecated]] attribute	N3760	c++14-lang	4.9	3.4	19.0*	4.9	15.0* 16.0	13.1.2*	5.14		8.6	16.1		
Sized deallocation	N3778	c++14-lang	5	3.4	19.0*	4.10.1	17.0		5.14		8.6	16.1		
Single quote as digit separator	N3781	c++14-lang	4.9	3.4	19.0*	4.10	16.0	13.1.2*	5.14		8.6	2015		
std::result_of and SFINAE	N3462	c++14	5	Yes	19.0*	N/A	N/A		5.15	Yes	8.6	N/A		
constexpr for <complex>	N3302	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
constexpr for <chrono>	N3469	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
constexpr for <array>	N3470	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
constexpr for <initializer_list>, <utility> and <tuple>	N3471	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
Improved std::integral_constant	N3545	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
User-defined literals for <chrono> and <string>	N3642	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
Null forward iterators	N3644	c++14	5*	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
std::quoted	N3654	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
Heterogeneous associative lookup	N3657	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
std::integer_sequence	N3658	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
std::shared_timed_mutex	N3659	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
std::exchange	N3668	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
Fixing constexpr member functions without const	N3669	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
std::get<T>()	N3670	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		
Dual-Range std::equal, std::is_permutation, std::mismatch	N3671	c++14	5	3.4	19.0*	N/A	N/A		5.15		8.6	N/A		

cppreference.com/w/cpp/compiler_support

C++17 feature	Paper(s)	Version	GCC	Clang	MSVC	EDG eccp	Intel C++	IBM XLC++	SunOracle C++	Embarcadero C++ Builder	Qnx	PortlandGroup (PGI)	HP aCC	Digital Mars C++
New auto rules for direct-list-initialization	N3922	c++17-lang	5	3.8	19.0*	4.10.1	17.0					17.7		
static_assert with no message	N3928	c++17-lang	6	2.9	19.10*	4.12	18.0					17.7		
typename in a template template parameter	N4051	c++17-lang	5	3.8	19.0*	4.10.1	17.0					17.7		
Removing trigraphs	N4086	c++17-lang	5	3.8	18.0*	5.0								
Nested namespace definition	N4230	c++17-lang	6	3.6	19.0*	4.12	17.0					17.7		
Attributes for namespaces and enumerators	N4266	c++17-lang	4.9 (namespaces) / 6 (enumerators)	3.6	19.0*	4.11	17.0					17.7		
u8 character literals	N4267	c++17-lang	6	3.6	19.0*	4.11	17.0					17.7		
Allow constant evaluation for all non-type template arguments	N4268	c++17-lang	6	3.6	19.12*	5.0								
Fold Expressions	N4295	c++17-lang	6	3.6	19.12*	4.14	19.0					18.1		
Remove Deprecated Use of the register Keyword	P0001R1	c++17-lang	7	3.8	19.11*	4.13	18.0					17.7		
Remove Deprecated operator++(bool)	P0002R1	c++17-lang	7	3.8	19.11*	4.13	18.0					17.7		
Removing Deprecated Excepton Specifications from C++17	P0003R5	c++17-lang	7	4	19.10*	4.14	19.0							
Make exception specifications part of the type system	P0012R1	c++17-lang	7	4	19.12*	4.14	19.0							
Aggregate initialization of classes with base classes	P0017R1	c++17-lang	7	3.9	19.14*	5.0								
Lambda capture of *this	P0018R3	c++17-lang	7	3.9	19.11*	4.14	19.0					18.1		
Using attribute namespaces without repetition	P0028R4	c++17-lang	7	3.9	19.13*	4.13	18.0					17.7		
Dynamic memory allocation for overaligned data	P0035R4	c++17-lang	7	4	19.12*	4.14								
Unary fold expressions and empty parameter packs	P0036R0	c++17-lang	6	3.9	19.12*	4.14								
__has_include in preprocessor conditionals	P0061R1	c++17-lang	5	Yes	19.11*	4.13	18.0					17.7		
Template argument deduction for class templates	P0091R3	c++17-lang	7	5	19.14*	5.0								
Non-type template parameters with auto type	P0127R2	c++17-lang	7	4	19.14*	5.0								
Guaranteed copy elision	P0135R1	c++17-lang	7	4	19.13*	5.0								
New specification for inheriting constructors (DR1941 et al)	P0136R1	c++17-lang	7	3.9	19.14*									
Replacement of class objects containing reference members	P0137R1	c++17-lang	7	6	19.14*	5.0								
Direct-list-initialization of enumerations	P0138R2	c++17-lang	7	3.8	19.11*	4.14	18.0							
Stricter expression evaluation order	P0145R3	c++17-lang	7	4	19.14*	5.0								
constexpr lambda expressions	P0170R1	c++17-lang	7	5	19.11*	4.14	19.0					18.1		
Differing begin and end types in range-based for	P0184R0	c++17-lang	6	3.9	19.10*	4.12	18.0					17.7		
[[fallthrough]] attribute	P0188R1	c++17-lang	7	3.9	19.10*	4.13	18.0					17.7		
[[nodiscard]] attribute	P0189R1	c++17-lang	7	3.9	19.11*	4.13	18.0					17.7		
Pack expansions in using-declarations	P0195R2	c++17-lang	7	4	19.14*	5.0								
[[maybe_unused]] attribute	P0212R1	c++17-lang	7	3.9	19.11*	4.13	18.0					17.7		
Structured Bindings	P0217R3	c++17-lang	7	4	19.11*	4.14	19.0					18.1		
Hexadecimal floating-point literals	P0245R1	c++17-lang	3.0	Yes	19.11*	4.13	18.0					17.7		
Ignore unknown attributes	P0283R2	c++17-lang	Yes	3.9	19.11*	4.13	18.0					17.7		
constexpr if statements	P0292R2	c++17-lang	7	3.9	19.11*	4.14	19.0					18.1		
Init-statements for if and switch	P0305R1	c++17-lang	7	3.9	19.11*	4.14	18.0					18.1		
Inline variables	P0386R2	c++17-lang	7	3.9*	19.12*	4.14	19.0					18.1		
DR: Matching of template template-arguments excludes compatible templates	P0522R0	c++17-lang	7	4	19.12*	5.0								
std::uncaught_exceptions()	N4259	c++17	6	3.7	19.0*	N/A	N/A					N/A		
Improving std::pair and std::tuple	N4387	c++17	6	4	19.0*	N/A	N/A					N/A		
std::any	P0220R1	c++17	7		19.12*	N/A	N/A					N/A		
std::variant	P0088R3	c++17	7	4	19.10*	N/A	N/A					N/A		
std::optional	P0220R1	c++17	7		19.10*	N/A	N/A					N/A		
std::shared_mutex (untimed)	N4508	c++17	6	3.7	19.0*	N/A	N/A					N/A		
std::string_view	N3921	c++17	7	4	19.10*	N/A	N/A					N/A		
Standardization of Parallelism TS	P0024R2	c++17			19.14*	N/A	18.0*					N/A		
Elementary string conversions	P0067R5	c++17	8 (partial)		19.14* (partial)	N/A	N/A					N/A		
Splicing Maps and Sets	P0083R3	c++17	7		19.12*	N/A	N/A					N/A		
Hardware interference size	P0154R1	c++17			19.11*	N/A	N/A					N/A		
std::filesystem	P0218R1	c++17	8	7	19.14*	N/A	N/A					N/A		
std::byte	P0298R3	c++17	7	5	19.11*	N/A	N/A					N/A		

Future

C++20, C++23, …

C++2x feature	Paper(s)	Version	GCC	Clang	MSVC	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	CAW	Portland Group (PGI)	HP aCC	Digital Mars C++
Concepts	P0734R0	c++2a-lang	6 (TS only)											
Contracts	P0542R5	c++2a-lang												
Default member initializers for bit-fields	P0683R1	c++2a-lang	8	6										
const&-qualified pointers to members	P0704R1	c++2a-lang	8	6										
Allow lambda-capture [=, this]	P0409R2	c++2a-lang	8	6										
__VA_OPT__	P0306R4	c++2a-lang	8 (partial*	6										
Designated initializers	P0329R4	c++2a-lang	4.7 (partial* 8	3.0 (partial*										
template-parameter-list for generic lambdas	P0428R2	c++2a-lang	8											
Initializer list constructors in class template argument deduction	P0702R1	c++2a-lang	8	6										
init-statements for range-based for	P0614R1	c++2a-lang	8											
Three-way comparison operator	P0515R0	c++2a-lang												
attribute [[no_unique_address]]	P0840R2	c++2a-lang												
attributes [[likely]] and [[unlikely]]	P0479R5	c++2a-lang												
typename optional	P0634R2	c++2a-lang												
Class types as non-type template parameters	P0732R2	c++2a-lang												
explicit(bool)	P0892R2	c++2a-lang												
constexpr virtual function	P1064R0	c++2a-lang												
efficient sized delete for variable sized classes	P0722R3	c++2a-lang												
integrating our feature-test macros	P0541R1	c++2a-lang												
Pack expansion in lambda init-capture	P0780R2	c++2a-lang												
Default constructible and assignable stateless lambdas	P0624R2	c++2a-lang	8											
Concept library	P0898R3	c++2a												
Three-way comparison operator support <compare>	P0515R3	c++2a												
std::endian	P0463R1	c++2a	8	7										
std::atomic_ref	P0019R8	c++2a												
std::bit_cast	P0476R2	c++2a												
Extending std::make_shared to support arrays	P0674R1	c++2a												
Calendar and timezone	P0355R7	c++2a												
std::span	P0122R7	c++2a												
Floating point atomic	P0020R6	c++2a												
Synchronized buffered ostream	P0053R7	c++2a												
constexpr for <algorithm> and <utility>	P0202R3	c++2a												
More constexpr for <complex>	P0415R1	c++2a												
std::remove_cvref	P0550R2	c++2a	8	6										
String prefix and suffix checking	P0457R2	c++2a		6										
Utility to convert a pointer to a raw pointer	P0653R2	c++2a	8	6										
<version>	P0754R2	c++2a	8	7										

- Contracts
- Concepts
- Ranges
- Modules
- Coroutines
- Even more constexpr
- Compile time reflection
- Metaclasses?
- Zero overhead exceptions?

zühlke
empowering ideas

HAM!

Thank you 

 Simplify C++! – www.arne-mertz.de

 @arne_mertz

 arne.mertz@zuehlke.com

emojis by EmojiOne