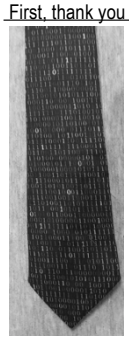
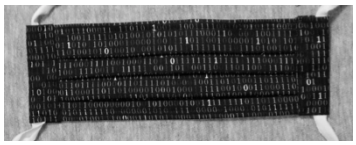
First, thank you to someone special





Copyright © 2020 by Walter E. Brown. All rights reserved.
1

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Edition: 2020-06-13. Copyright © 2020 by Walter E. Brown. All rights reserved.

A little about me

- B.A. (math's); M.S., Ph.D. (computer science).
- Professional programmer for over 50 years, programming in C++ since 1982.
- Experienced in industry, academia, consulting, and research:
  - Founded a Computer Science Dept.; served as Professor and Dept. Head; taught and mentored at all levels.
  - Managed and mentored the programming staff for a reseller.
  - Lectured internationally as a software consultant and commercial trainer.
  - Retired from the Scientific Computing Division at Fermilab, specializing in C++ programming and in-house consulting.
- Not dead — still doing training & consulting. (Email me!)

Copyright © 2020 by Walter E. Brown. All rights reserved.
3

Emeritus participant in C++ standardization

- Written ~170 papers for WG21, proposing such now-standard C++ library features as gcd/lcm, cbegin/cend, common_type, and void_t, as well as all of headers <random> and <ratio>.
- Influenced such core language features as *alias templates*, *contextual conversions*, and *variable templates*; recently worked on *requires-expressions*, operator<=>, and more!
- Conceived and served as Project Editor for *Int'l Standard on Mathematical Special Functions in C++* (ISO/IEC 29124), now incorporated into C++17's <cmath>.
- Be forewarned: Based on my training and experience, I hold some rather strong opinions about computer software and programming methodology — these opinions are not shared by all programmers, but they should be! ☺

Copyright © 2020 by Walter E. Brown. All rights reserved.
4

The origin(?) of "Lightning Talks"

- "A *lightning talk* is a very short presentation … given at a conference or similar forum."
- It's claimed that:
  - "The term was first coined … in June 2000 [although] …
  - "The practice of lightning talks was first known to be used in 1997."
- Yet I remember (from ~50 years ago) that:
  - At the 1st programmers' conference I ever attended …
  - We had an hour of short talks under the common title, "**What Every FORTRAN Programmer Ought to Know**."

Copyright © 2020 by Walter E. Brown. All rights reserved.
5

Today: an homage

- That 1971 talk was memorable (at least to me):
  - In part because not every topic needs a full hour, right?
- So I have for you today several mini-talks:
  - In the approximate style that I witnessed ~50 years ago.
  - With C++ (rather than Fortran) as their unifying theme.
- As the overall title for this talk, I first thought of:
  - "**What Every C++ Programmer Ought to Know**."
  - But that seemed too obvious; too much like a clone.
  - Instead, I chose the simpler "**Lightning Strikes!**"

Copyright © 2020 by Walter E. Brown. All rights reserved.
6

Where will lightning strike today?

Topics

- **Why are they named *Lambdas*?**
- **Underlying principles of C++ declarations**
- **Classifying the universe of C++ types**
- **What C++20 owes to a 20th century mathematician**

Copyright © 2020 by Walter E. Brown. All rights reserved. 7

α? β? ζ? θ? μ? δ?

Why Are They Named *Lambdas*?

σ? γ? π? ω? ψ?

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Edition: 2020-06-10. Copyright © 2020 by Walter E. Brown. All rights reserved.

We owe much to these pioneers of computation …

Alan Turing
1912–1954

Emil Post
1897–1954

Alonzo Church
1903–1995

Kurt Gödel
1906–1978

Stephen Kleene
1909–1994

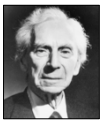
Copyright © 2020 by Walter E. Brown. All rights reserved. 11

… As well as to these pioneers of computation

Andrey Markov, Jr.
1903–1979

Axel Thue
1863–1922

Dana Scott
1932–



Bertrand Russell
1872–1970

Alfred Whitehead
1861–1947

Copyright © 2020 by Walter E. Brown. All rights reserved. 12

From theory to practice

- They researched the foundations of mathematics:
  - Eventually their respective works largely converged into a branch of math's known as computability theory.
  - And Church's notation became the foundation of the LISP programming language (McCarthy, 1958).
- "Lisp is today's equivalent of Latin. Educated people are supposed to have studied and forgotten it."

— Peter Dimov,
Simple C++11 metaprogramming,
2015

Copyright © 2020 by Walter E. Brown. All rights reserved. 13

Functional programming lives on

- In such higher-order programming, functions are not only callable, but are copyable values, too:
  - In C++, function objects have always fit this description, …
  - So C++11 introduced "lambda-expr's [as] a concise way to create a simple function object" (see *[expr.prim.lambda]/1*).
  - And, as we know, C++ lambdas have evolved ever since.
- Some consider that "the untyped λ-calculus was the first object-oriented language."

— William R. Cook,
On Understanding Data Abstraction, Revisited,
OOPSLA 2009

Copyright © 2020 by Walter E. Brown. All rights reserved. 14

Origin of lambda as a term of art

- Today, math functions are named (*e.g.*, *f*(y) = 3y + 1):
  - (It's believed Leonhard Euler was first to do so [1734].)
- But Alonzo Church used (1930s) anonymous fctn def'ns (*e.g.*, **ŷ** . 3y + 1) in his computing foundations work:
  - He'd adapted the **ŷ** notation (for function-abstraction) from Whitehead & Russell's notation (for class-abstraction).
  - (BTW, that's the form I first learned in graduate school.)
- Church wrote (1964) that when he published his work:
  - His caret (hat, **^**) was typeset separately (*e.g.*, **^y** . 3y + 1) …
  - Thus resembling a Greek capital Lambda, that was later changed to lower case (*e.g.*, **λy** . 3y + 1).

Copyright © 2020 by Walter E. Brown. All rights reserved. 15

However, …

- Dana Scott (Church's former student and 1976 Turing Award recipient) told a different story in his [2018] talk *Looking Backward; Looking Forward*.
- Scott said that he once appealed to Church's son-in-law (John Addison) to ask Church the origin of lambda:
  - So Addison wrote Church a postcard, asking …
  - "… Why did you choose lambda as your operator?"
- Church allegedly returned the postcard in an envelope, having annotated the postcard with the words, "eenie, meenie, miney, mo"!

Copyright © 2020 by Walter E. Brown. All rights reserved. 16

Whether typographical or arbitrary …

- Church's notation for functions/function abstraction became known as lambda expressions, …
- And Church's collected work in this area became known as the lambda calculus, …
- Long since recognized as a universal model of computation …
- That's been proven equivalent in computational power to the models set forth by Post, Kleene, and Turing, among others.

Copyright © 2020 by Walter E. Brown. All rights reserved. 17

Why Are They Named *Lambdas*?

FIN

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Copyright © 2019, 2020 by Walter E. Brown. All rights reserved.

Underlying Principles of C++ Declaration Syntax

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Edition: 2020-06-13. Copyright © 2020 by Walter E. Brown. All rights reserved.

C declarations' guiding principle

"The style used in expressions carries through to declarations…."

— Dennis M. Ritchie, PH.D.,
The Development of the C Language
(presented at HOPL-II, 1993)

Copyright © 2020 by Walter E. Brown. All rights reserved. 22

From expressions to declarations

- That principle applies in C++, too — let's look at some examples.
- For an entity x, what expressions could give a result of type int?
  - If a simple variable: x.
  - If an array: x[···].
  - If a pointer: *x.
  - If a function: x(···).
- So, how would x be declared in each of these cases?
  - int x; *// simple variable*
  - int x[···]; *// array*
  - int *x; *// pointer*
  - int x(···); *// function*

Copyright © 2020 by Walter E. Brown. All rights reserved.

Parts of a declaration, left to right

- It starts with a type, followed by …
- An expression (declarator) of a form that could give a value of that type, and then ...
- Punctuation (a semicolon if at end, else a comma).
- Examples (redux):
  - int x; *// simple variable*
  - int x[···]; *// array*
  - int *x; *// pointer*
  - int x(···); *// function*

Copyright © 2020 by Walter E. Brown. All rights reserved.

Precedence applies, even in declarators

- *E.g.*, recall that operator ( ) has precedence over (binds more tightly than/has seniority over) operator * :
  - So in an expression, operator ( ) claims all its operands …
  - Before operator * can begin to claim its operands.
- Similar expressions can therefore differ in meaning:
  - * p(···) *// p is callable, and …*
    // … the call's result is dereferenceable
  - (*q)(···) *// q is dereferenceable, and …*
    // … the dereferenced result is callable
- That difference is reflected in their resp. declarators:
  - int * p(···); *// p has a function type*
  - int (*q)(···); *// q has a pointer-to-function type*

Copyright © 2020 by Walter E. Brown. All rights reserved.

Multiple declarators

- Multiple declarators w/ a common type may share it:
  - int k, a[10], *p, g(double);
  - int k *// known as the One True Comma layout*
    , a[10]
    , *p
    , g(double)
    ;
- Even class members can be declared to share a type:
  - struct S {
    int zero = 0, one[1] = { 1 }, two( ); *// not recommended*
    };
  - int S :: two( ) { return 2; }

Copyright © 2020 by Walter E. Brown. All rights reserved.

More syntax options

- We earlier said: "It starts with a type, …":
  - We did **not** say, "It starts with a type name, ….!
- Examples:
  - struct S { ··· } s, t;
    // defines type S, also defines variables s and t
  - struct { ··· } u;
    // defines u with an anonymous type
- There seem few good reasons to exploit this form.

Copyright © 2020 by Walter E. Brown. All rights reserved.

C++ offers still more syntax options

- Attributes ([[···]]).
- Initializers (= ···, (···), { ··· }).
- Qualifiers (const, volatile, &, &&).
- Specifiers (constexpr, consteval, constinit, extern, friend, inline, mutable, static, thread_local, typedef, virtual, …).

Copyright © 2020 by Walter E. Brown. All rights reserved.

Recap of the main takeaway

"The style used in expressions carries through to declarations…."

— Dennis M. Ritchie, Ph.D.,
The Development of the C Language
(presented at HOPL-II, 1993)

Copyright © 2020 by Walter E. Brown. All rights reserved. 29

Outside the C subset, C++ had to adapt

- Recall that C has no reference types, and that C++ has no expressions of reference type:
  - "If an expression initially has the type *reference to T*, the type is adjusted to *T* …" (see *[expr.type]/1*).
  - So no C++ operator can produce any ref. type.
- Instead, C++ adapted pointer syntax to declare an entity of reference type:
  - We just use & or && in the declarator instead of ∗.
  - Example: `int ∗ x ··· ;` *// x has pointer type*
    compare to: `int & y ··· ;` *// y has lvalue-ref type*
    and to: `int && z ··· ;` *// z has rvalue-ref type*

Copyright © 2020 by Walter E. Brown. All rights reserved. 30

One more C++ innovation was needed …

- Because C has no scope-resolution operator (::).
- Example:
  - `struct S { bool b; double d( float ); };`
  - Data member S :: b has object type bool, while ...
  - Member fctn S :: d has fctn type double(float).
- So a pointer-to-member-of indicates the member-of class and the member's type:
  - `bool S :: * pb = & S :: b;`
  - `double ( S :: * pd ) ( float ) = & S :: d;`

Copyright © 2020 by Walter E. Brown. All rights reserved. 31

A final caveat

- This is, of course, still not the complete story.
- For example, we haven't mentioned type aliases:
  - How they are declared, and …
  - Their role in declaring other entities.
- Nor have we mentioned, *e.g.*, the forbidden ref-to-ref (a.k.a. reference-collapsing) types.
- So, as seems typical in C++:
  - There's always more to learn!

Copyright © 2020 by Walter E. Brown. All rights reserved. 32

Underlying Principles of C++ Declaration Syntax

FIN

Walter E. Brown, Ph.D.
< webrown.cpp @ gmail.com >

Copyright © 2020 by Walter E. Brown. All rights reserved.

Walter E. Brown, Ph.D.
< webrown.cpp @ gmail.com >

Edition: 2020-06-13. Copyright © 2019, 2020 by Walter E. Brown. All rights reserved.

Let's explore/partition this universe

- The C++ *primary* type classifications do not overlap:
  - The core language specifies these in *[basic.types]*.
  - The standard library specifies corresponding type traits in *[meta.unary]*.
  - ☺ Fortunately, they mostly agree.
- A type's *cv*-qual's does not affect its classification.
- But here's a related puzzle for you (answer at the end):

> For which primary classification(s) of types T would the following predicate yield false?
>
> std::is_const_v< T const >

Copyright © 2020 by Walter E. Brown. All rights reserved.

Fundamental types

- void *types*
  - There are 4, counting *cv*-qualification.
- std::nullptr_t *types*:
  - Like the voids, these 4 have their own classification.
- *Arithmetic types*:
  - *Floating-point types*: { float, double, long double }
  - *Integral/integer types*: See next page.

Copyright © 2020 by Walter E. Brown. All rights reserved.

Integral types, a.k.a. *integer types*

- { bool, char, wchar_t, char8_t, char16_t, char32_t }
- *Signed integer types*:
  - *Standard signed integer types*:
    signed { char, short int, int, long int, long long int }
  - *Extended signed integer types*:
    Implementation-defined (but does anyone?).
- *Unsigned integer types*:
  - *Standard unsigned integer types*:
    unsigned { char, short int, int, long int, long long int }
  - *Extended unsigned integer types*:
    Implementation-defined (again, does anyone?).

Copyright © 2020 by Walter E. Brown. All rights reserved.

Newsflash!

- Earlier this year, WG14 (the C committee) decided:
  - To deprecate the intmax_t type, along with …
  - All macros, functions, *etc*., related to it.
- So there seems to be a resolution underway:
  - To address at least some of the long-standing issues …
  - Associated with the *Extended {un}signed integer types*.

Copyright © 2020 by Walter E. Brown. All rights reserved.

Compound types (*i.e.*, types based on $n \geq 1$ other types)

- *Arrays of* { *known, unknown* } *extent*:
  - Composed of objects of a single specified type.
- { *Lvalue, rvalue* } *references*:
  - To an { object, free function, static member function } of some specified type.
- *Pointers*:
  - To void, or …
  - To an { object, free function, static member function } of some specified type.

Copyright © 2020 by Walter E. Brown. All rights reserved.

Compound types (continued)

- *Functions*:
  - Have $m \geq 0$ parameters of specified types, and …
  - Return { void, object, reference } type.
- *Classes*:
  - Contain a sequence of $m \geq 0$ objects of various specified types, and …
  - Contain a set of specified types, enumerations, and functions for manipulating these objects, and …
  - Contain a set of restrictions on these entities' access.

Copyright © 2020 by Walter E. Brown. All rights reserved.

Compound types (concluded)

- *Unions*:
  - Classes that may contain objects of different specified types ...
  - At different times.
- *Enumerations*:
  - Comprise a set of named constant values …
  - That share a single underlying specified integral type.
- *Pointers-to-member-of*:
  - Identify non-static { data, function } members of a specified type …
  - Within objects of a specified class type.

Copyright © 2020 by Walter E. Brown. All rights reserved. 44

Some composite type classifications

- *Class types*:
  - All types declared with a *class-key* { class, struct, union }.
- *Function object types*:
  - All ptr-to-function types, and …
  - All class types with an operator( ) member, and …
  - All class types with a conversion function whose target type is a { ptr, ref, ref-to-ptr }-to-function type.
- *Callable types*:
  - All { function object, ptr-to-member } types.

Copyright © 2020 by Walter E. Brown. All rights reserved. 45

A few more composite type classifications

- *Scalar types*:
  - All { arithmetic, enumeration, pointer, pointer-to-member, std::nullptr_t } types.
- *Object types*:
  - All non-{ function, reference, void } types.
- *Incomplete types*:
  - All void types, and …
  - All declared-but-not-defined class types, and …
  - All enumerations "in certain contexts", and …
  - All array types such that { bound is unknown, element type is incomplete }.

Copyright © 2020 by Walter E. Brown. All rights reserved. 46

Example: deciding a type's completeness (since C++11)

- A trait employing (a variant of) the *detection idiom*:
  - template< class, class = std::size_t > *// primary template*
    struct is_complete : std::false_type { };
  - template< class T > *// partial specialization*
    struct is_complete<T, decltype(sizeof(T)) >
    : std::true_type { };
- Use this trait carefully! For some types, the answer might be different at different points in your program:
  - *E.g.*, false after a class's forward declaration (incomplete) vs. true after that class's later definition (now complete).
  - Such inconsistency violates C++'s One Definition Rule (the ODR), so instantiate this trait only once per type.

Copyright © 2020 by Walter E. Brown. All rights reserved. 47

Finally: the answer to the puzzle

- std::is_const_v< ***T*** const > yields false when …
  - T is either a *reference* type or a *function* type.
- Perhaps this is not useful knowledge every day, but:
  - Historically, the std::is_function trait was implemented via a primary template + 48 (!) partial specializations:
    - 48 = 4 *cv* qual's × 3 ref qual's × 2 noexcept × 2 ellipsis arg.
    - These don't include non-standard calling conventions!
  - template< class T >
    struct is_function
    : bool_value< not is_const_v< T const > *// per bullet at top*
    and not is_reference_v< T > *// rule out ref types*
    > { };

Copyright © 2020 by Walter E. Brown. All rights reserved. 48

The Universe of C++ Types

FIN

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Copyright © 2019, 2020 by Walter E. Brown. All rights reserved.

What Does C++20 Owe to

Prof. Dr. Emmy Noether?

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Edition: 2020-06-10. Copyright © 2020 by Walter E. Brown. All rights reserved.

Emmy Noether (1882–1935)

adapted from A. Borschel-Dan

- Although she was a mathematician's daughter, she was not allowed to enroll in Univ. classes:
  - The academic senate warned that coed education would "overthrow all academic order." Said a faculty member, "What will our soldiers think when they … find that they are required to learn at the feet of a woman?"
  - To which mathematician David Hilbert famously replied, "I do not see that the candidate's gender is an argument against her admission…. After all, we are a university, not a bathhouse."

- Noether was at last officially permitted to study, but only by asking permission from each lecturer.

Copyright © 2020 by Walter E. Brown. All rights reserved.

At the start of her career

- After winning the right to matriculate, Noether earned a Ph.D. in 1907, but no faculty jobs were then open to women:
  - So she worked without pay for 7+ years.
  - Then "hired" in 1915 by David Hilbert at the Univ. of Göttingen, but spent years lecturing under his name, still unpaid.
  - Finally appointed *Lehrbeauftragte für Algebra* in 1923, Noether had to flee to the U.S.A. in 1933.

- She "produced papers and theories at a staggering pace," including many rarely acknowledged contributions to works written by her students and colleagues.

Copyright © 2020 by Walter E. Brown. All rights reserved.

Some of Noether's significant contributions

- "The development of abstract algebra, … one of the most distinctive innovations of 20th century math's, is largely due to her – in published papers, in lectures, and in … influence on her contemporaries."

  — Nathan Jacobson

- *Noether's theorem* [1915, 1918] is "one of the most important mathematical theorems ever proved in guiding the development of modern physics, possibly on a par with the Pythagorean theorem."

  — Leon Lederman & Christopher Hill

Copyright © 2020 by Walter E. Brown. All rights reserved.

"Emmy Noether saved General Relativity" w/ her theorem

- "[Since] 1915, General Relativity, a new … way of thinking about gravity, had captured the attention of physicists ….
- "But [this new theory had] difficulties which even Einstein could not resolve.
- "We would likely not be celebrating this landmark theory were it not for a mathematician who, at her prime, couldn't even secure a teaching role in her homeland because of her gender.
- "Noether's were the mathematical breakthroughs that general relativity needed to win over physicists."

  — Robert Lea

Copyright © 2020 by Walter E. Brown. All rights reserved.

But Noether is now recalled by relatively few

- Her mathematical originality was "absolute beyond comparison"

  — Bartel L. van der Waerden

- She "changed the face of algebra by her work."

  — Hermann Weyl

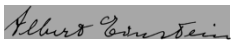
- She "is ... the greatest woman mathematician who has ever lived; and the greatest woman scientist of any sort now living, and a scholar at least on the plane of Madame Curie."

  — Norbert Wiener

Copyright © 2020 by Walter E. Brown. All rights reserved.

A very special posthumous tribute (excerpted)

- "Within the past few days a distinguished mathematician, Professor Emmy Noether, … died in her fifty-third year.
- "In the judgment of the most competent living mathematicians, [she] was the most significant creative mathematical genius thus far produced … ."



— Albert Einstein,
"Professor Einstein Writes in Appreciation of a Fellow-Mathematician,"
The New York Times, 1935-05-05

Copyright © 2020 by Walter E. Brown. All rights reserved. 58

A biographer summarizes

- "In her 53 years,
  - "many spent bucking a system that impeded her pursuit of mathematics,
  - "Noether had an extraordinary impact on both algebra and physics.
- "There's no telling what else she might have accomplished if society and fate had been more kind."

— Steve Nadis,
The Universe According to Emmy Noether

Copyright © 2020 by Walter E. Brown. All rights reserved. 59

What does Noether's work have to do with C++20?

- "It was she who taught us to think in terms of simple and general algebraic concepts…"

— Pavel S. Alexandrov

- "For Emmy Noether, relationships among numbers, functions, and operations became … amenable to generalisation … only after they have been … reduced to general conceptual relationships…."

— Bartel L. van der Waerden

Copyright © 2020 by Walter E. Brown. All rights reserved. 60

"The greatest thing … in 20th century mathematics!"

- "What was [her] idea? [It was that y]ou could separate concepts from implementation. You could just deal with concepts….
- "So, if anybody invented generic programming, she did…. She realized it all. She realized that you could fully separate these two things.
- "Her contribution needs to be remembered."

— Alexander Stepanov, 2003

Copyright © 2020 by Walter E. Brown. All rights reserved. 61

C++20 Owes Concepts to

Prof. Dr. **Emmy Noether**

R.I.P.

"Her contribution needs to be remembered."

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Copyright © 2020 by Walter E. Brown. All rights reserved.

Lightning Strikes!

FIN

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

Copyright © 2020 by Walter E. Brown. All rights reserved.