The Universe of C++ Types

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >

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!)

Emeritus participant in C++ standardization

- Written >160 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! ☺

Let's explore 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.
- *cv*-qualification does not affect a type's classification:
  - But here's a related puzzle for you, namely …
  - For which primary classification(s) of types ***T*** **would the** following predicate yield false?

    std::is_const_v< ***T*** **const >**

Fundamental types

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

Integral (or *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.

Newsflash!

- New this week: a relevant C (WG14) proposal.
- Title: "intmax_t, a way out: Ease the definition of extended integer types":
  - Author: J. Gustedt
  - Date: 2019-09-13
  - WG14 document #: N2425 v2
- So we at least one have one proposed solution for the issues surrounding *Extended unsigned integer types*.

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

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

Compound types (continued)

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

Compound types (concluded)

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

Some 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 classes, and …
  - All enumerations "in certain contexts", and …
  - All arrays whose bound is unknown or whose element type is incomplete.

Yet more 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 types, and …
  - All ptr-to-member types.

Example: deciding a type's completeness (C++17)

- Apply (a variant of) the *detection idiom*:
  - template< class, class = std::size_t > *// primary template*
    struct is_complete : std::false_type { };
  - template< class T > *// explicit specialization*
    struct is_complete<T, decltype(sizeof(T)) >
    : std::true_type { };
- Use carefully! For some compound 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 is problematic, so don't ask twice.

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 not useful knowledge every day, but:
  - Historically, the std::is_function trait was implemented via a primary template + 48 (!) partial specializations.
  - Doesn't include non-standard calling conventions!
  - template< class T >
    struct is_function
    : std::bool_value< not (is_const_v< T const >
    or is_reference_v< T >)
    > { };

The Universe of C++ Types

FIN

WALTER E. BROWN, PH.D.

< webrown.cpp @ gmail.com >