The C++20 Standard Library - Beyond Ranges

Jeff Garland

Created: 2019-09-18 Wed 06:36

intro

talk goals

- io – formatted output [~15 min]
  - output format `{fmt}`
  - `sync_buf` and `osyncstream`
  - efficient `stringbuf` and `stringstream` access
- container and algorithm updates [~12 min]
  - `span`
  - associative containers `contains`
  - uniform container erase algorithms
  - `to_array` helper (unlikely)
  - erase returns on `list`
  - `shift` algorithms (unlikely)

talk goals II

- chrono [~8 min]
- concurrency: threading and atomics [~10 min]
  - `jthread` and `stop_token`
  - `atomic for shared_ptr ~weak_ptr`
  - `atomic_ref<T>`

talk goals lll (really fast)

- misc [~5 min]
  - string `starts_with` `ends_with`
  - `shared_ptr`, `unique_ptr` initializtion
  - math
  - `[[nodiscard]]` on functions
- bit manipulation [~5 min]
  - rotation and count
  - power functions
  - `bit_cast`
  - endian

c++20 library changes not discussed today

- ranges is the big one
- concept support for ranges
- 3way compare support (aka spaceship support)
- concurrency
  - coordination types: `latch` and `barrier`
  - `counting_semaphore` and `binary_semaphore`
- `char8_t`, `u8string` and `u8string_view`

c++20 library changes not discussed II

- `constexpr` on algorithms and containers
- various `noexcept` additions
- `variant` changes
  - sane variant converting constructor
    http://wg21.link/P0608
  - visit<R> explicit return type for visit
    http://wg21.link/P0655
- algorithm vectorization policies
  http://wg21.link/P1001

things I will and won’t do in talk

- **will:** take questions as we go - until we get behind
- will: defer questions I can't answer immediately
- will: no doubt get something wrong

things I will and won't do in talk

- will: show you lots of code
- will: shorten namespaces (`std::ranges`) and leave out `#includes` on slides
- wont: show you an example that hasn't compiled
  - in some universe with some set of libraries/compilers (unless noted)
  - the environment : Linux g++ 8.2, 8.3, 9
    - v9 - trunk as of mid-Mar 2019
    - typically with -fconcepts and -stdc++20
    - various referenced library implementations

status of c++20 implementations

“Patience you must have my young padawan”

- *Yoda*

c++20 Library implementation status

- c++20 is feature complete
- g++, clang, Visual C++

  https://en.cppreference.com/w/cpp/compiler_support

- current working paper http://wg21.link/n4830

input-output

- output format `{fmt}`
- `syncbuf` and `osyncstream`
- efficient `stringbuf` and `stringstream` access

output string format `std::format`

- python style formatting to c++
- positional parameters
- custom type formats
- faster and less overhead then iostreams
- integrated with chrono (output only)
- in header `<format>`

hello world

```
#include <fmt/core.h> //will be <format>

{
  //hello world
  string s = fmt::format("{} there {}\n", "hello", "world");
  cout << s;  //hello there world
}
```

- format function replaces the {} with corresponding parameters
- parameters can be any type
  - all built in numbers and strings support out of box
  - custom types can be supported

indexed parameters

```
{
  //indexed parameters -- parm[0] == 42, parm[1] == "hello"
  //format reorders in string
  string s = fmt::format("{1} and {0}\n", 42, "hello");
  cout << s;  //hello and 42
}
```

- cannot mix indexed and non-indexed

```
string s = fmt::format("{} and {0}\n", 42, "hello"); //error
```

simple escaping

```
{
  //escaping '{' and '}'
  string s = fmt::format("{{ untouched  }} hello {}\n", 42);
  cout << s;  //{ untouched  } hello 42
}
```

crazy town

```
{
  //escaping '{' and '}'
  string s = fmt::format("{{ in brace {} }}\n", 42);
  cout << s;  //{ in brace 42 }
}
{
  //braceapoloza
  string s = fmt::format("{{{{{}}}}}\n", 42);
  cout << s;  //{{42}}
}
```

floating point format

```
{
 int width = 10;
 int precision = 3;
 // format like {0:10.3f}
 auto s = fmt::format("{0:{1}.{2}f}", 12.345678, width, precision);
 // s == "    12.346"
}
```

rich language for fill, alignment, conversion

```
string s0 = format("{:6}", 42);   // s0 is "    42"
string s1 = format("{:6}", 'x');   // s1 is "x     "
string s2 = format("{:*<6}", 'x'); // s2 is "x*****"
string s3 = format("{:*>6}", 'x'); // s3 is "*****x"
string s4 = format("{:*^6}", 'x'); // s4 is "**x***"
string s5 = format("{:6}", true); // s5 is "true  "

string s6 = format("{0:b} {0:d} {0:o}", 42); // s6 is "101010 42 52"
string s7 = format("{0:#x} {0:#X}", 42);     // s7 is "0x2a 0X2A"
```

custom type usage

```
//annotated version
enum class my_color {red, green, blue, orange};

my_color c = my_color::red;
string s = fmt::format("{}\n", c); //please work!
```

how to extend for custom type

```
//annotated version
enum class my_color {red, green, blue, orange};

namespace fmt {  // <- template specialiation in fmt namespace
  template <>    // <-  syntax for specialization
  struct formatter<my_color>: formatter<string_view> { //inherit from f
     template <typename FormatContext>
     auto format(my_color c, FormatContext &ctx) {
         string_view name = "unknown color"; // <- some default text
         switch (c) {
           case my_color::red:   name = "red"; break;
           case my_color::green: name = "green"; break;
           case my_color::blue:  name = "blue"; break;
         } //should there be a fallthru instead?
       return formatter<string_view>::format(name, ctx);
     }
  }; //struct formatter<my_color>
}//namespace fmt
```

fmt diagnostics/safety

- currently runtime diagnostic (std::exception derivative)
- cases should be checkable in future
- what causes exceptions?
  - wrong parameter count
  - mismatched parameter types
- advice
  - dont use in catch blocks
  - dont use in destructors
  - dont use in non-exception friendly locations

mis-matched parameter count

```
//2 format parameters only one actual parameter - throws
string s = fmt::format("{1} hello {0}\n", "world");
```

wrong format typing causes error, even when convertible

```
{
 //no automatic conversion like python
 //print("Sammy ate {0:f} percent of a {1}!".format(75, "pizza"))
 //Sammy ate 75.000000 percent of a pizza!

    string s = fmt::format("float and decimal: {1:f} and {0:d}\n",
                           42, 42);
    cout << s;

 //terminate called after throwing an instance of 'fmt::v5::format_erro
 //what():  invalid type specifier
 //formatting numbers

}
```

- https://www.digitalocean.com/community/tutorials/ho to-use-string-formatters-in-python-3

`fmt` references and status

- moved by LWG in Cologne
- http://wg21.link/p0645
- https://github.com/fmtlib/fmt

`syncbuf` and `osyncstream` overview

- streams in multi-threaded application can produce garbled output
- `osyncstream` used as an automatic-duration variable
- buffers output operations for a wrapped stream
- transfers sync buffer stream to underlying on destruction
- header `<syncstream>`

`osyncstream` simple example

```
 #include <syncstream> //new header
//...
{
  osyncstream buffered_out(cout);
  buffered_output << " the answer might be 42" << endl;
} //on destruction buffered_out now calls emit()
  //emit() flushes the buffer to cout
{
  std::ofstream out_file("my_file");
  osyncstream buffered_out(out_file);
  std::emit_on_flush(buffered_out);
  buffered_out << "hello world" << endl; //calls emit()
}
```

`osyncstream` thread example

```
void do_output(std::ostream& ofile ) {
  osyncstream buf_file{ofile};
  //   ...complex code that writes to file and may throw
} //ensures buf_file is flushed to outfile

std::ofstream out_file("my_file");
std::thread   out_thread1( &do_output, out_file )
std::thread   out_thread2( &do_output, out_file )
```

`syncbuf` and `osyncstream` synopsis

```
template <class charT,
          class traits = char_traits<charT>,
          class Allocator = allocator<charT>>
class basic_syncbuf;

using syncbuf  = basic_syncbuf<char>;
using wsyncbuf = basic_syncbuf<wchar_t>;

template <class charT,
          class traits = char_traits<charT>,
          class Allocator = allocator<charT> >
class basic_osyncstream;

using osyncstream = basic_osyncstream<char>;
using wosyncstream = basic_osyncstream<wchar_t>;
```

`syncbuf` and `osyncstream` references

- in working draft
- paper http://wg21.link/p0053
- https://github.com/PeterSommerlad/SC22WG21_Pap

`stringstream` and `basic_stringbuf` updates

- `basic_stringbuf` used in `stringstream`
- changes to allow for control of buffer allocator
- output interfaces to provide `string_view` from `stringstream`

`stringstream` to `string_view` example

```
#include <sstream>
//...
std::stringstream s;
s << "put some data into the stream";

std::string_view sv = s.view();
```

`stringstream` and `basic_stringbuf` resources

- http://wg21.link/p0408
- implementation https://github.com/PeterSommerlad/SC22WG21_Pap

container and algorithm updates

- `span`
- associative containers `contains`
- uniform container erase algorithms
- `to_array` helper
- erase returns on `list`
- `shift` algorithms

`span`

- span is a non-owning 'view' over contiguous sequence of object
- cheap to copy - implementation is a pointer and size
- constant time complexity for all member functions
- defined in header `<span>`
- unlike most 'view types' can mutate
- constexpr ready

span construction

```
vector<int> vi = { 1, 2, 3, 4, 5 };
span<int>   si ( vi );

array<int, 5> ai = { 1, 2, 3, 4, 5 };
span<int>     si2 ( ai );

int cai[] = { 1, 2, 3, 4, 5 };
span<int>     si3( cai );

array<string, 2> sa = { "world" , "hello "};
span<string> ss ( sa );
```

span as a function parameter

```
void print_reverse(span<int> si) { //by value
  for ( auto i  : std::ranges::reverse_view{si} ) {
    cout << i << " " ;
  }
  cout << "\n";
}

int main () {

  vector<int> vi = { 1, 2, 3, 4, 5 };
  print_reverse( vi ); //5 4 3 2 1

  int cai[] = { 1, 2, 3, 4, 5 };
  print_reverse ( cai ); //5 4 3 2 1

  span<int> si ( vi );
  print_reverse( si.first(2) ); //2 1
  print_reverse( si.last(2) );  //5 4
```

basic public accessors

- supports collection-like interfaces (`begin`, `end`, `operator[]`)
- `data` pointer to start of sequence
- `size` number of elements
- `size_bytes` memory size
- `empty` true if no elements

static and dynamic extent

```
array<int, 5> ai = { 1, 2, 3, 4, 5 };
span<int, 5> si5( ai );
print_reverse ( si5 ); //5 4 3 2 1

span<int, 5> si3( ai );
```

span resources

- in the working paper
- http://wg21.link/P0122
- implementation https://github.com/tcbrindle/span
- should span be regular http://wg21.link/p1085

`contains` method for associative containers

```
 std::map<string, SomeType> m = initMap();
 if (m.find( "foo" ) != m.end()) {
//or
 if (m.contains( "foo" )) {  //uh huh -- easier to read
```

`contains` method signatures

```
 //adds member contains (2 overloads) to map, multimap,
 //set, and multiset

 //check with the key type
bool contains(const key_type& x) const;

//check with a type that is comparable to key_type
template <class K> bool contains(const K& x) const;
```

`contains` method signatures for unordered

```
 // adds member contains to unordered_map, unordered_multimap,
 // unordered_set, and unordered_multiset

 //check with the key type
bool contains(const key_type& x) const;
```

`contains` reference

- paper: http://wg21.link/p0458
- status: in working draft

uniform container erasure

- motivation - removing elements unreasonably difficult
- when there's and idiom, thats a problem
- simplify user code
- Erase idiom

```
std::vector<SomeType> c = ...;
c.erase(remove_if(c.begin(), c.end(), pred), c.end());
```

`erase` and `erase_if` signatures

```
template<class T, class Allocator, class U>
void erase(list<T, Allocator>& c, const U& value);

Effects: Equivalent to:
   erase_if(c, [&](auto& elem) return elem == value; );

template<class T, class Allocator, class Predicate>
void erase_if(list<T, Allocator>& c, Predicate pred);

Effects: Equivalent to: c.remove_if(pred);
```

`erase` and `erase_if` scope

- both apply to `basic_string`, `deque`, `forward_list`, `list`, `vector`
- `erase_if` applies to `map`, `multimap`, `set`, `multiset`, `unordered_*`
- since associative containers already have `erase( keytype )` functions

uniform container erase status and references

- merged in working draft
- starting paper http://wg21.link/n4009
- San Diego final http://wg21.link/p1209
- lifted from library fundamentals v2
- in vcc since 2015

`to_array` helper

motivating example

```
// array<char, 4> from array{ "foo" } ? no
// because: array{ "foo", "bar" } creates array<char const*, 2>

auto a1 = to_array("foo");          // array<char, 4>
auto a2 = array{ "foo" };           // array<char const*, 1>
```

- from the LTFSv2 with updates

`to_array` references

- http://wg21.link/P0325.html
- status: LWG moved in Cologne

return values on `remove`

- impacts `list` and `forwardlist`
- why throw away information?
- change `remove()`, `removeif()` and `unique()`
- return `container::sizetype` of number elements removed
- makes consistent with `map`/`set` `erase` function

return values status and reference

- in working draft
- http://wg21.link/p0646

`shift_left` and `shift_right` algorithms

- move elements a fixed amount
- return an iterator to one past shift point
- `size` of sequence unchanged

`shift_left` example

```
vector<int>    vi = { 1, 2, 3, 4, 5 };

auto itr = shift_left(vi.begin(), vi.end(), 2);
//vi = 3 4 5 4 5
//           ^
//          itr
vi.erase( itr, vi.end());

for ( auto i : vi ) {
  cout << i << " ";
}
cout << "\n";  //3 4 5
```

`shift` synopsis

```
// 25.6.14, shift
template<class ForwardIterator>
constexpr ForwardIterator
shift_left(ForwardIterator first, ForwardIterator last,
           typename iterator_traits<ForwardIterator>::difference_type r

template<class ForwardIterator>
constexpr ForwardIterator
shift_right(ForwardIterator first, ForwardIterator last,
            typename iterator_traits<ForwardIterator>::difference_type

 //also overloads for parallel execution
```

`shift` reference

- merged in working draft
- http://wg21.link/p0769
- https://github.com/danra/shift_proposal
- will be ranges versions (not merged in c++20)

chrono

- chrono fundamentals
- date support features
- i/o

chrono design fundamentals

- extends existing capabilites to calendric functions
- two primary types in chrono
  - structure/field types like `year_month_day`
  - calculation types like `sys_days`, `time_point`, `seconds`
- field types are used more for i/o and conversions
- calculation types - small memory, fast compuations
- supports constexpr

simple calculations

```
year_month_day ymd(year(2019)/05/07);
sys_days d( ymd );
d += weeks(1);
cout << ymd << "/n"; //2019-05-07
cout << d << "/n";   //2019-05-14
```

chrono error handling

- no exceptions
- the 'ok' method tells you if a value is good

```
year_month_day ymd(year(2019)/100/200);
if ( !ymd.ok() ) {
  cout << "bad ymd\n"; //executes
}
```

chrono i/o

- stream-based input/output
- `std::format` based ouput for strings
- uses `sprintf` style format strings

chrono format output

```
//cant compile this...
const std::chrono::year_month_day ymd(2019_y/05/07);
std::string s = std::format("the date is {:%m-%d-%Y}.\n", ymd);
cout << s << "\n";  //
```

optional locale aware format output

```
//not compiled
auto zt = std::chrono::zoned_time(...);
std::cout << std::format(std::locale{"fi_FI"},
                         "Localized time is {:%c}\n", zt);
```

other c++20 chrono features

- leap seconds
- IANA timezone database support
- localized times
- many other types `year_month`, `gps_clock`, `weekday_indexed`

chrono references and status

- reference implementation https://github.com/HowardHinnant/date
- chrono for c++20 http://wg21.link/p0355 (in working paper)
- std::format and chrono http://wg21.link/p1361 (LWG moved in Cologne)
- chrono format strings https://en.cppreference.com/w/cpp/chrono/format

concurrency: threading and atomics

- `jthread` and `stop_token`
- `atomic<shared_ptr>`, `atomic<weak_ptr>`
- `atomic_ref<T>`

`jthread` and `stop_token`

- `jthread` (joining thread) is an update for `std::thread`
- automatically joins in the destructor
- `stop_token` provides cooperative shutdown
- in header `<thread>`

thread gone wrong?

```
#include <thread>

void do_nothing()
{
  this_thread::sleep_for( seconds( 1 ) );
}

int
main()
{
  cout << "start" << endl;
  {

    std::thread t(do_nothing);

  } // what bad thing happens here?
}
```

`jthread` instead

```
#include <thread>

{

  std::jthread t(do_nothing);

} //I'm cool if you're cool


// fine, because...
~jthread()
{
   if( joinable() ) {
     request_stop();
     join();
   }
}
```

- `jthread` defaults to joining

`jthread` cooperative shutdown

```
class jthread {
public:
  [[nodiscard]] stop_source get_stop_source() noexcept;
  [[nodiscard]] stop_token  get_stop_token()  const noexcept;
  bool request_stop() noexcept;
....
```

- 3 types `stop_source`, `stop_token`, `stop_callback`

`jthread` cooperative shutdown

```
// user code -- (not compiled)
{
  std::jthread t([]( std::stop_token token ){
                    while ( !token.requested_stop())
                    {
                       //do stuff
                    }
                 }
        );

}
```

new waiting interfaces with `stop_token`

```
// 32.6.4.2 interruptible waits:

template<class Lock, class Predicate>
bool
wait_until(Lock& lock,Predicate pred,stop_token stoken);

template<class Lock, class Clock, class Duration, class Predicate>
bool
wait_until(Lock& lock,
           const chrono::time_point<Clock, Duration>& abs_time,
           Predicate pred,
           stop_token stoken);

template<class Lock, class Rep, class Period, class Predicate>
bool
wait_for(Lock& lock,
          const chrono::duration<Rep, Period>& rel_time,
          Predicate pred,
          stop_token stoken)
```

`jthread` references and status

- motivation http://wg21.link/p0660r0
- wording http://wg21.link/p0660
- implementation https://github.com/josuttis/jthread

atomic `shared_ptr` and `weak_ptr`

- `atomic_shared_ptr` interfaces are error prone
- removes `atomic_shared_ptr<T>` and `atomic_weak_ptr<T>` alias templates
- adds new atomic interface

```
// 23.11.3, atomic smart pointers
template <class T> struct atomic<shared_ptr<T>>;
template <class T> struct atomic<weak_ptr<T>>;
```

- reference http://wg21.link/p0718

`atomic_ref`

- High performance computing motivates
- like `atomic<T>` but for reference to non-atomic
- specializations for integral types (char, int, long, etc)
- specializations for floating point types
- `atomic_ref` http://wg21.link/P0019

atomic_ref example

```
vector<int> vi { 1, 2, 3, 4 }; //not atomics

for ( int i=0; i != vi.end(); i++) {
    value = atomic_ref<int>{ vi[i] };
    value += 1;  //atomic fetch_add
}
```

misc

- `shared_ptr`, `unique_ptr` default initialization
- string `starts_with` `ends_with`
- `source_location`
- math
- `[[nodiscard]]` on functions

`shared_ptr`, `unique_ptr` default initialization

- All about performance
- arrays of built in types often initialized after creation
- hence initialization performed by `make_shared` and `make_unique` redundant

default pointer init example

```
//shared_ptr to a default-initialized double[1024]
//where each element has an indeterminate value
shared_ptr<double[]> p = make_shared_default_init<double[]>(1024);
```

default initialization for `unique_ptr` and `shared_ptr` reference

- Smart pointer creation with default initialization http://wg21.link/p1020
- http://boost.org/libs/smart_ptr [since 1.56]

`starts_with` and `ends_with` for `string` and `string_view`

```
//string_view
constexpr bool starts_with(basic_string_view x) const noexcept;
constexpr bool starts_with(charT x) const noexcept;
constexpr bool starts_with(const charT* x) const;
constexpr bool ends_with(basic_string_view x) const noexcept;
constexpr bool ends_with(charT x) const noexcept;
constexpr bool ends_with(const charT* x) const;
```

- prefix and suffix checks
- http://wg21.link/p0457

`source_location`

- modern replacement for FILE and LINE C macros
- more sophisticated abilities
- new header `<source_location>`
- originally planned for use in contracts and stacktrace

source_location synopis

```
//[source_location.syn]
struct source_location {

   // source location construction
   static consteval source_location current() noexcept;
   constexpr source_location() noexcept;

   // source location field access
   constexpr uint_least32_t line() const noexcept;
   constexpr uint_least32_t column() const noexcept;
   constexpr const char* file_name() const noexcept;
   constexpr const char* function_name() const noexcept;
```

source_location usage

```
void f(source_location a = source_location::current()) {
  // values in b refer to the line below
  source_location b = source_location::current();
  // ...
}

// f's first argument corresponds to this line of code
f();

// f's first argument gets the same values as c, above
source_location c = source_location::current();
f(c);
```

source_location usage

```
struct s {
   source_location contruct_loc = source_location::current();
   int other_member;

   // values of construct_loc refer to the location of the calling func
   s(source_location loc = source_location::current())
    : construct_loc(loc)
   {}

   // values of construct_loc refer to this location
   s(int i) :
     other_member( i )
   {}

   // values of construct_loc refer to this location
   s(double)
   {}
};
```

`source_location` resources

- http://wg21.link/p1208
- source location implementations - not aware of one

math constants

- new header `<math>`
- new namespace `std::math`
- provided for various floating point types
- nearest representable value

math constants example

```
inline constexpr long double el = e_v<long double>;
inline constexpr long double log2el = log2e_v<long double>;
inline constexpr long double log10el = log10e_v<long double>;
inline constexpr long double pil = pi_v<long double>;
inline constexpr long double inv_pil = inv_pi_v<long double>;
inline constexpr long double inv_sqrtpil = inv_sqrtpi_v<long double>;
inline constexpr long double ln2l = ln2_v<long double>;
inline constexpr long double ln10l = ln10_v<long double>;
inline constexpr long double sqrt2l = sqrt2_v<long double>;
inline constexpr long double sqrt3l = sqrt3_v<long double>;
inline constexpr long double inv_sqrt3l = inv_sqrt3_v<long double>;
inline constexpr long double egammal = egamma_v<long double>;
inline constexpr long double phil = phi_v<long double>;
```

math constants reference and status

- pre-reviewed for Cologne
- paper http://wg21.link/p0631
- boost http://boost.org/libs/math

midpoint

- why is midpoint difficult?
- for integers, overflow is the primary problem
- for floating point it's rounding
- same issue for pointer type

```
template<typename A>
constexpr A midpoint(A a, A b) noexcept;
```

linear interpolation (lerp)

- handy for many domains
- why is linear interpolation difficult?
- really easy to get it wrong for non-mathematicians

```
//from P0811 -- wanted properties
exactness: lerp(a,b,0)==a && lerp(a,b,1)==b
monotonicity: cmp(lerp(a,b,t2),
                  lerp(a,b,t1)) * cmp(t2,t1) * cmp(b,a) >= 0,
       where cmp is an arithmetic three-way comparison function
determinacy: result of NaN only for lerp(a,a,INFINITY)
boundedness: t<0 || t>1 || isfinite(lerp(a,b,t))
consistency: lerp(a,a,t)==a
```

- interface

```
constexpr float lerp(float a, float b, float t);
constexpr double lerp(double a, double b, double t);
constexpr long double lerp(long double a, long double b, long double
```

midpoint/lerp references and status

- midpoint and lerp are in p0811
- http://wg21.link/p0811
- in working draft

c++20 will add nodiscard to many functions

- including `empty`, `allocate`, `new`
- http://wg21.link/p0600r1

bit manipulation functions

- rotation and count
- power functions
- `bit_cast`
- endian

header `<bit>` overview

- new in c++20
- function templates to access, manipulate, and process individual bits and bit sequences.

power functions

- Defined in header `<bit>` in namespace std
- `ispow2` checks if a number is an integral power of two
- `ceil2` finds the smallest integral power of two not less than the given value
- `floor2` finds the largest integral power of two not greater than the given value
- `log2p1` finds the smallest number of bits needed to represent the given value

bit rotation and counting

```
//Header bit
// 23.20.2, rotating
template<class T>
  [[nodiscard]] constexpr T rotl(T x, int s) noexcept;
template<class T>
  [[nodiscard]] constexpr T rotr(T x, int s) noexcept;

// 23.20.3, counting
template<class T>
  constexpr int countl_zero(T x) noexcept;
template<class T>
  constexpr int countl_one(T x) noexcept;
template<class T>
  constexpr int countr_zero(T x) noexcept;
template<class T>
  constexpr int countr_one(T x) noexcept;
template<class T>
  constexpr int popcount(T x) noexcept;
}
```

bit manipulation status and reference

- low level routines
- expect instruction level optimizations
- reference http://wg21.link/p0553
- earlier paper http://wg21.link/n3864

bit_cast

- low level facility to cast from type to type
- `reinterpret_cast` and `union` can be error prone
- size of the two types must match

```
// 25.5.3, bit_cast
//reinterpret the object representation of one type as that of anoth
template<typename To, typename From>
constexpr To bit_cast(const From& from) noexcept;
```

- http://wg21.link/p0476
- in working draft

endian

```
//short description: detect the byte order at compile or runtime
//header: <bit>

  enum class endian {
    little = /*implementation-defined*/,
    big    = /*implementation-defined*/,
    native = /*implementation-defined*/
  };
```

compile time usage

```
class my_byte_dependent_type
{
 public:
  static constexpr std::endian endian = std::endian::native;
```

runtime usage

```
if (endian::native == endian::big)
    // do big endian encoding
else if (endian::native == endian::little)
    // handle little endian
else
    // handle mixed endian (mythical unicorn)
```

endian status and reference

- merged in working draft
- Howard H. http://wg21.link/p0463
- boost endian http://www.boost.org/libs/endian
- other bit manipulation facilities
  - std::bitset (since 1998)
  - boost dynamic bitset http://www.boost.org/libs/dynamic_bitset

observations

- library updates for c++20 are big
- some key building blocks
- many small additions
- overall: nicer cleaner more capable code

Thanks!!

"Design is nothing if not decision making."

Henri Petroski 2007 - “Small Things Considered: Why There Is No Perfect Design”