C++17 with examples

What’s in C++17

- The Parallelism TS, a.k.a. “Parallel STL.”
- The Library Fundamentals 1 TS
- The File System TS (based on boost.filesystem)
- The Mathematical Special Functions IS (based on boost.math)
- Miscellaneous

What’s in C++17

- **The Parallelism TS, a.k.a. “Parallel STL.”**

Execution policy - std::parallel::

- Sequential_execution_policy - a parallel algorithm's execution may **not** be parallelized.
- Parallel_execution_policy - a parallel algorithm's execution may be parallelized
- Parallel_vector_execution_policy - a parallel algorithm's execution may be vectorized and parallelized.
- [Dynamic] execution_policy - dynamic control over standard algorithm execution

Execution policy - std::parallel::

```
constexpr sequential_execution_policy        seq{};
constexpr parallel_execution_policy          par{};
constexpr parallel_vector_execution_policy   par_vec{};
```

Because different parallel architectures may require idiosyncratic parameters for efficient execution, implementations of the Standard Library may provide additional execution policies to those described in this Technical Specification as extensions.

Exception handling

- If memory resources are not available - std::bad_alloc
- parallel_vector_execution_policy => std::terminate
- Sequential_execution_policy and parallel_execution_policy the algorithm exits throwing exception_list containing all uncaught exception
- If the execution policy is of any other type the behavior is implementation-defined

Table 2 — Table of parallel algorithms

adjacent_difference	adjacent_find	all_of	any_of
copy	copy_if	copy_n	count
count_if	equal	exclusive_scan	fill
fill_n	find	find_end	find_first_of
find_if	find_if_not	for_each	for_each_n
generate	generate_n	includes	inclusive_scan
inner_product	inplace_merge	is_heap	is_heap_until
is_partitioned	is_sorted	is_sorted_until	lexicographical_compare
max_element	merge	min_element	minmax_element
mismatch	move	none_of	nth_element
partial_sort	partial_sort_copy	partition	partition_copy
reduce	remove	remove_copy	remove_copy_if
remove_if	replace	replace_copy	replace_copy_if
replace_if	reverse	reverse_copy	rotate
rotate_copy	search	search_n	set_difference
set_intersection	set_symmetric_difference	set_union	sort
stable_partition	stable_sort	swap_ranges	transform
transform_exclusive_scan	transform_inclusive_scan	transform_reduce	uninitialized_copy
uninitialized_copy_n	uninitialized_fill	uninitialized_fill_n	unique
unique_copy			

[*Note:* Not all algorithms in the Standard Library have counterparts in Table 2. — *end note*]

Parallelism TS Algorithms

```
for_each( std::par,  first, last, [](auto x){ process(x); });

// explicitly sequential sort

sort(sequential, v.begin(), v.end());

// permitting parallel execution

sort(par, v.begin(), v.end());

// permitting vectorization as well

sort(par_vec, v.begin(), v.end());
```

Parallel TS Algorithms

It is the caller's responsibility to ensure correctness, for example that the invocation does not introduce data races or deadlocks.

Parallel TS Algorithms

```
int a[] = {0,1};

std::vector<int> v;

for_each(par, std::begin(a), std::end(a), [&](int i) {
  v.push_back(i*2+1);
});
```

Data race!

Parallel TS Algorithms

```
int x=0;
std::mutex m;
int a[] = {1,2};
for_each(par, std::begin(a), std::end(a), [&](int) {
  m.lock();
  ++x;
  m.unlock();
});
```

Task Block

Task Block proposal

a library function template **define_task_block** and a library class **task_block** with member functions **run** and **wait** that together enable developers to write expressive and portable fork-join parallel code.

Task Block

```
template <typename Func>
int traverse(node& n, Func && compute)
{
    int left = 0, right = 0;
    define_task_block(
        [&](task_block<>& tr) {
            if (n.left)
                tr.run([&] { left = traverse(*n.left, compute); });
            if (n.right)
                tr.run([&] { right = traverse(*n.right, compute); });
        });

    return compute(n) + left + right;
}
```

What’s in C++17

- The Parallelism TS, a.k.a. “Parallel STL.”
- **The Library Fundamentals 1 TS**

The Library Fundamentals 1 TS

```
// Calling a function with a tuple of arguments

template <class F, class Tuple>

constexpr decltype(auto) apply(F&& f, Tuple&& t);
```

The Library Fundamentals 1 TS

- Metaprogramming and type traits (50-100+ броя):

template <class T>

constexpr bool **is_lvalue_reference_v** = is_lvalue_reference<T>::value;

- Compile-time rational arithmetic

template <class R1, class R2>

constexpr bool **ratio_not_equal_v** = ratio_not_equal<R1, R2>::value;

Optional objects

An optional object for object types is an object that contains the storage for another object and manages the lifetime of this contained object, if any.

The contained object may be initialized after the optional object has been initialized, and may be destroyed before the optional object has been destroyed. The initialization state of the contained object is tracked by the optional object.

A common use case for optional is the return value of a function that may fail. As opposed to other approaches, such as **std::pair<T,bool>**

optional

```
optional<char> get_async_input()
{
    if ( !queue.empty() )
        return optional<char>(queue.top());
    else return optional<char>(); // uninitialized
}
```

any

An object of class any stores an instance of any type that satisfies the constructor requirements or is empty, and this is referred to as the state of the class any object. The stored instance is called the contained object. Two states are equivalent if they are either both empty or if both are not empty and if the contained objects are equivalent.

The non-member any_cast functions provide type-safe access to the contained object.

any

```
std::any a = 1;
std::cout << std::any_cast<int>(a) << '\n';
a = 3.14;
std::cout << std::any_cast<double>(a) << '\n';
a = true;
std::cout << std::boolalpha << std::any_cast<bool>(a) << '\n';
```

string_view

The class template basic_string_view describes an object that can refer to a **constant** contiguous sequence of char-like objects with the first element of the sequence at position zero.

A typical implementation holds only two members: a pointer to constant CharT and a size.

std::search

```
template<typename Container>
bool in_quote(const Container& cont, const std::string& s)
{
    return std::search(cont.begin(), cont.end(), s.begin(), s.end()) != cont.end();
}
```

std::sample

```
int main()
{
    std::string in = "abcdefgh", out;
    std::sample(in.begin(), in.end(), std::back_inserter(out),
                5, std::mt19937{std::random_device{}()});
    std::cout << "five random letters out of " << in << " : " << out << '\n';
}
```

What’s in C++17

- The Parallelism TS, a.k.a. “Parallel STL.”
- The Library Fundamentals 1 TS
- **The File System TS (based on boost.filesystem)**

The File System TS

```
namespace fs = std::filesystem;

// fail to copy directory
 fs::create_directory("sandbox/abc");
 try {
     fs::copy_file("sandbox/abc", "sandbox/def");
 } catch(fs::filesystem_error& e) {
     std::cout << "Could not copy sandbox/abc: " << e.what() << '\n';
 }
 fs::remove_all("sandbox");
```

```
int main()
{
     std::filesystem::create_directories("sandbox/a/b");
     std::ofstream("sandbox/file1.txt");
     for (auto& p : std::filesystem::recursive_directory_iterator("sandbox"))
          std::cout << p << '\n';
     std::filesystem::remove_all("sandbox");
}
Output:
sandbox\a
sandbox\a\b
sandbox\file1.txt
```

What's in C++17

- The Parallelism TS, a.k.a. "Parallel STL."
- The Library Fundamentals 1 TS
- The File System TS (based on boost.filesystem)
- **The Mathematical Special Functions IS (based on boost.math)**

The Mathematical Special Functions

bessel_I
bessel_J
bessel_K
bessel_j
beta
ei

ellint_E
ellint_E2
ellint_F
ellint_K
ellint_P
ellint_P2

hermite
hyperg_1F1
hyperg_2F1
laguerre_0

laguerre_m
legendre_Pl
legendre_Plm
neumann_N
neumann_n
sph_Y

zeta

What's in C++17

- The Parallelism TS, a.k.a. "Parallel STL."
- The Library Fundamentals 1 TS
- The File System TS (based on boost.filesystem)
- The Mathematical Special Functions IS (based on boost.math)
- **Miscellaneous**

Miscellaneous

- Lambdas are now allowed inside constexpr functions
- Lambdas can now capture a copy of *this object by value, using the notation **[*this].**
- The range-for loop can now deal with generalized ranges where the “end” type is different from the “begin” type
- **[[fallthrough]]**, **[[nodiscard]]**, **[[maybe_unused]]** attributes
- Hexadecimal floating-point literals
- and more

Range-based for loop

```
{
    auto && __range = range_expression ;
    for (auto __begin = begin_expr, __end = end_expr;__begin != __end; ++__begin)
    {
        range_declaration = *__begin;
        Loop_statement
    }
}
```

Range-based for loop

```
{
    auto && __range = range_expression ;
    auto __begin = begin_expr ; // __range.begin() or begin(__range)
    auto __end = end_expr ;     // __range.end() or end(__range)
    for ( ; __begin != __end; ++__begin)
    {
        range_declaration = *__begin;
        Loop_statement
    }
}
```

Probably in C++17 during June meeting

- **if constexpr** to allow branches that are evaluated at compile time.
- **Template parameter deduction for constructors** - pair p(2, 3.5); instead of pair<int,double> p(2, 4.5)
- **Defining the order of expression evaluation**
- **operator.** (dot)
- **Defaulted comparisons**, to generate ==, !=, <, <=, >, >= for types that don't write them by hand.

Bibliography

- Parallelism TS
- C++ Extensions for Library Fundamentals
- File System TS
- Special mathematical functions