# C++20 STL Features: 1 Year of Development on GitHub

Stephan T. Lavavej "Steh-fin Lah-wah-wade"

Principal Software Engineer, Visual C++ Libraries

stl@microsoft.com

@StephanTLavavej

## Getting Started

- Please hold your questions until the end
  - Write down the slide numbers

- Part 0: Overview
  - What's happened in the last year
- Part 1: C++20 STL Features
  - Everything here is Standard, except as noted
- Part 2: GitHub Development
  - For contributors and observers

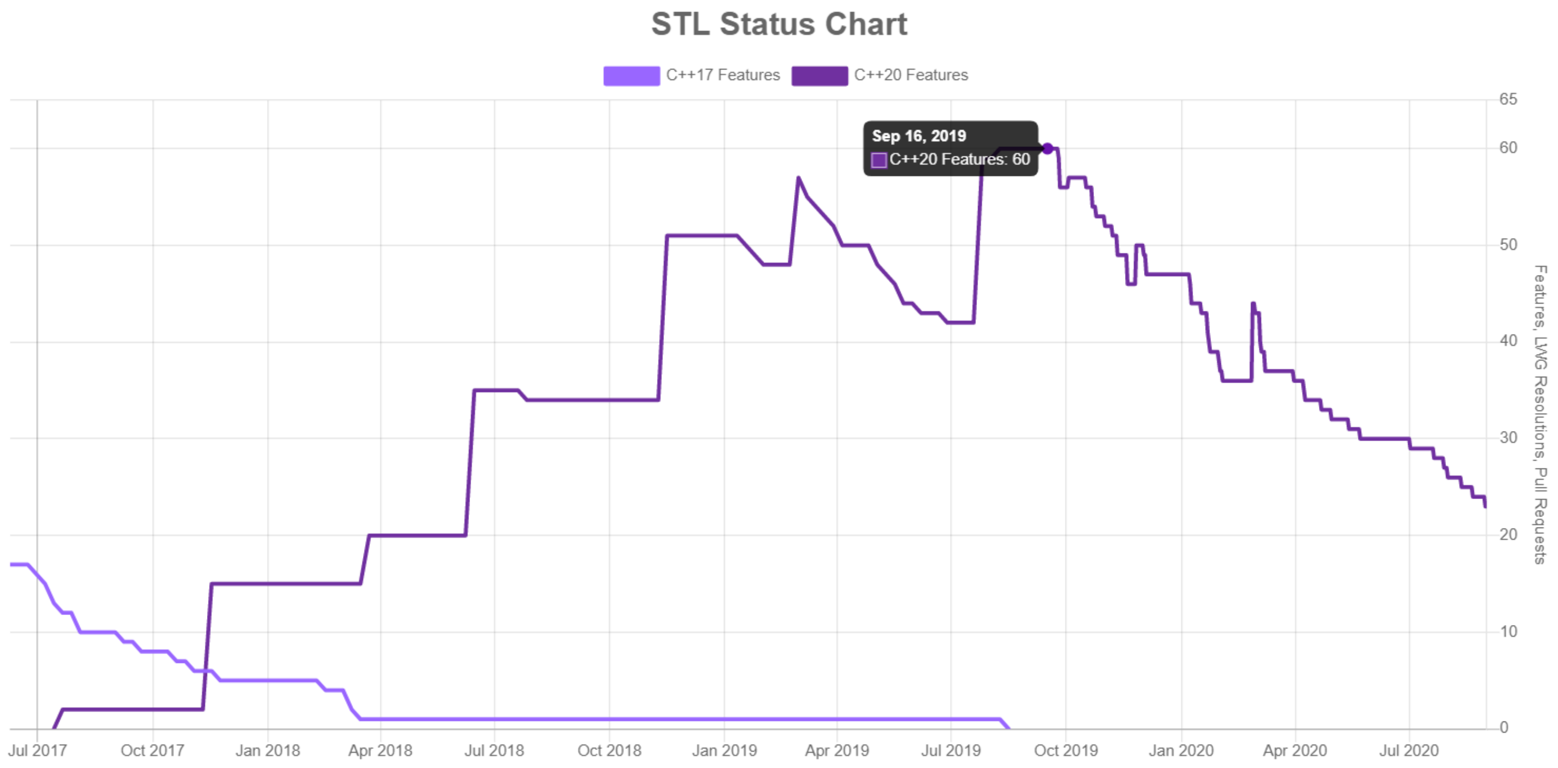
## Overview

Part 0

## CppCon 2019 -> CppCon 2020

- Announced at CppCon 2019
  - `github.com/microsoft/STL`
  - Apache License v2.0 with LLVM Exception
- Implemented ~50 C++20 features
  - Majority from our amazing contributors
  - Extensively reviewed and tested
- Released VS 2019 16.5, 16.6, 16.7, 16.8 Preview 3
- GitHub migration ongoing
  - Build system (native desktop), test suite, issue database
- Goal: Complete C++20 in 2020
  - No promises, but we're working hard; refers to repo only

# C++20 Features: 60 -> 23 Left



# C++20 STL Features

Part 1

## Comparing Integers

## Usual Arithmetic Conversions

```
#include <iostream>
using namespace std;
int main() {
    cout << boolalpha;
    short s          = -1;
    unsigned int ui = 1729;
    cout << (s < ui) << endl;
}
```

- What does this print?
  - `warning C4018: '<': signed/unsigned mismatch`
  - `warning: comparison of integers of different signs: 'short' and 'unsigned int' [-Wsign-compare]`
  - This prints `false` 😱

# Integer Comparison Functions

- GH-621 implemented by Neargye

```
#include <iostream>
#include <utility>
using namespace std;
int main() {
    cout << boolalpha;
    short s            = -1;
    unsigned int ui = 1729;
    cout << cmp_less(s, ui) << endl;
}
```

- This prints `true` ✓

# <utility>

```
template <class T, class U>
    constexpr bool cmp_equal(T t, U u) noexcept;
template <class T, class U>
    constexpr bool cmp_not_equal(T t, U u) noexcept;
template <class T, class U>
    constexpr bool cmp_less(T t, U u) noexcept;
template <class T, class U>
    constexpr bool cmp_greater(T t, U u) noexcept;
template <class T, class U>
    constexpr bool cmp_less_equal(T t, U u) noexcept;
template <class T, class U>
    constexpr bool cmp_greater_equal(T t, U u) noexcept;
template <class R, class T>
    constexpr bool in_range(T t) noexcept;
```

## constexpr Algorithms

# C++20 constexpr Everything

- `constexpr` enables compile-time programming without arcane templates
- Algorithms:
  - P0202R3 `constexpr <algorithm>` And `exchange()` (GH-425)
  - P0879R0 `constexpr` Algorithms, Part II (GH-425, BillyONeal)
  - P1645R1 `constexpr <numeric>` Algorithms (GH-399, Neargye)
- More:
  - P0415R1 `constexpr <complex>` Again (GH-367, Neargye)
  - P0595R2 `is_constant_evaluated()` (GH-353, Jennifer Yao – MSVC compiler)
  - P1006R1 `constexpr pointer_to()` (GH-397, AdamBucior)
  - P1023R0 `constexpr array` Comparisons (GH-599, Weheineman)
  - P1032R1 Miscellaneous `constexpr` (GH-491, miscco)
  - P1065R2 `constexpr` INVOKE (GH-703, AdamBucior)
- Work in progress (mnatsuhara):
  - GH-37 P0784R7 Library Support For More `constexpr` Containers
  - GH-43 P0980R1 `constexpr string`
  - GH-45 P1004R2 `constexpr vector`

## `constexpr` In Everyday Programming

- `constexpr` has steadily increased in power
  - It's just code that runs on your machine instead of the user's
- Example: lookup tables
  - Everyone uses lookup tables
  - Arrays are maximally space-efficient, very time-efficient
  - Sorted arrays: `binary_search`, `lower_bound`, `equal_range`
- Now you can use `constexpr` algorithms
  - Easy: `static_assert` with `constexpr is_sorted`
  - `string_view` is also `constexpr`; not limited to numeric data
  - More advanced: `constexpr sort` your lookup tables

# static_assert is_sorted

- Compile-time evaluation = no run-time codegen, even in debug builds

```
#include <algorithm>
#include <array>
#include <filesystem>
#include <string>
#include <string_view>
using namespace std;
int main() {
    static constexpr array skipped_extensions{".dll"sv, ".exe"sv, ".obj"sv};
    static_assert(ranges::is_sorted(skipped_extensions));
    for (const auto& ent : filesystem::recursive_directory_iterator{"."}) {
        const string extension = ent.path().extension().string();
        if (ranges::binary_search(skipped_extensions, extension)) {
            continue;
        }
        // ...
    }
}
```

## Erasing Elements

## Container Erasure Strategies

- Erasing unwanted elements before C++20:
  - `vector`-like: `erase`-`remove` idiom
  - `list`-like: `remove`/`remove_if` member functions
  - `map`-like: handwritten loop calling `m.erase(iter)`
- Many potential hazards 🙀
  - `remove_if(v.begin(), v.end(), pred);` 🐞
  - `v.erase(remove_if(v.begin(), v.end(), pred));` 🐞
  - Quadratic complexity `vec.erase(iter)` loop 🐌
  - Invalidating iterators while looping 🐞
  - Skipping elements while looping 🐞

# Uniform Container Erasure

- GH-236 and GH-566 implemented by SuperWig

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
void print(const vector<string>& v) {
    for (const auto& e : v) { cout << e << " "; }
    cout << "\n";
}
int main() {
    vector<string> v{"bear", "dog", "cat", "lion", "ox", "dog"};
    auto pred = [](const auto& s) { return s.size() > 3; };
    erase_if(v, pred);
    print(v); // dog cat ox dog
    erase(v, "dog");
    print(v); // cat ox
}
```

## atomic_ref

# atomic vs. atomic_ref

- `atomic<T>`: atomic access and storage for `T`
- `atomic_ref<T>`: atomic access for separate `T`
- Scenarios (see P0019R8):
  - Distinct single-threaded and multi-threaded parts
    - Pay the cost for atomic operations only when necessary
  - Distinct non-conflicting and conflicting parts
  - Layout compatibility; works with `T` within C `structs`
- Near-identical interface
  - Encapsulates platform/architecture-specific code
  - `atomic`/`atomic_ref` support `wait`/`notify_one`/`notify_all`

# Highly Artificial Example

- GH-843 implemented by AlexGuteniev

```
#include <atomic>
#include <iostream>
#include <thread>
using namespace std;
int main() {
   int i{500};
   i += 500; // ordinary read/write
   { atomic_ref atom{i};
     thread t1{[&atom] { for (int val{0}, x{0}; x < 70;) {
       if (atom.compare_exchange_weak(val, val + 10)) { ++x; }}}};
     thread t2{[&atom] { for (int val{0}, y{0}; y < 29;) {
       if (atom.compare_exchange_weak(val, val + 1)) { ++y; }}}};
     t1.join(); t2.join(); }
   cout << i << endl; // ordinary read, 1729
}
```

## span

## span Encapsulates Ptr-Len Pairs

- `span` is a non-owning view of contiguous elements
- `span<T>` is like a pair of `T*` and `size_t`, but better
  - `span`s are implicitly constructible from arrays and `vector`s
  - Avoids ownership confusion, accidental pointer arithmetic
  - `span<T>` $\boxed{\rightarrow}$ `span<const T>`; `span<Derived>` ✕ `span<Base>`
- Shallow `const`, like `T*`
  - `span<const T>` can be reassigned, can't modify elements
  - `const span<T>` can't be reassigned, can modify elements
- MSVC's `std::span` checks debug; `gsl::span` always

## Contiguous Range Constructor

- GH-142, GH-500, GH-587 implemented by miscco

```
#include <array>
#include <iostream>
#include <span>
#include <vector>
using namespace std;
void print(const span<const int> s) { // not a template!
    for (const auto& e : s) { cout << e << " "; }
    cout << endl;
}
int main() {
    static constexpr int classic[]{1, 2, 3};
    print(classic);
    print(array{4, 5, 6});
    print(vector{7, 8, 9});
}
```

# Contiguous > Random-Access

- Can we `print(deque{-1, -1, -1});` ?
  - `candidate function not viable: no known conversion from 'std::deque<int, std::allocator<int> >' to 'const span<const int>' for 1st argument`
- This constructor is powered by C++20 ranges!
  - Specifically `ranges::contiguous_range` (and more)
  - Extensible to user-defined contiguous ranges

## And More!

## So Many New Features, Including:

- `atomic<shared_ptr<T>>`, `atomic<weak_ptr<T>>`
  - GH-601 by AdamBucior
- `<bit>` (`bit_cast`, rotating/counting, power-of-2)
  - Several PRs by barcharcraz
- `make_shared()` For Arrays
  - GH-309 by AdamBucior, tested by Weheineman
- `midpoint()`, `lerp()`
  - Paolo Torres and BillyONeal initially implemented this
  - GH-1048 by statementreply completed this
- `<numbers>` Math Constants
  - GH-261 by SuperWig

# GitHub Development

Part 2

## How We Use GitHub

- Code: linear history, few feature branches
- Issues: cxx20, LWG, bug, performance, etc.
- Pull Requests: Used for all development
- Continuous Integration: Azure Pipelines
  - Code format validation: clang-format, line length, etc.
- Projects: Code reviews, C++20 features, etc.
- Wiki: Changelog, checklists, other resources
- Status Chart: Generated via GitHub's GraphQL API

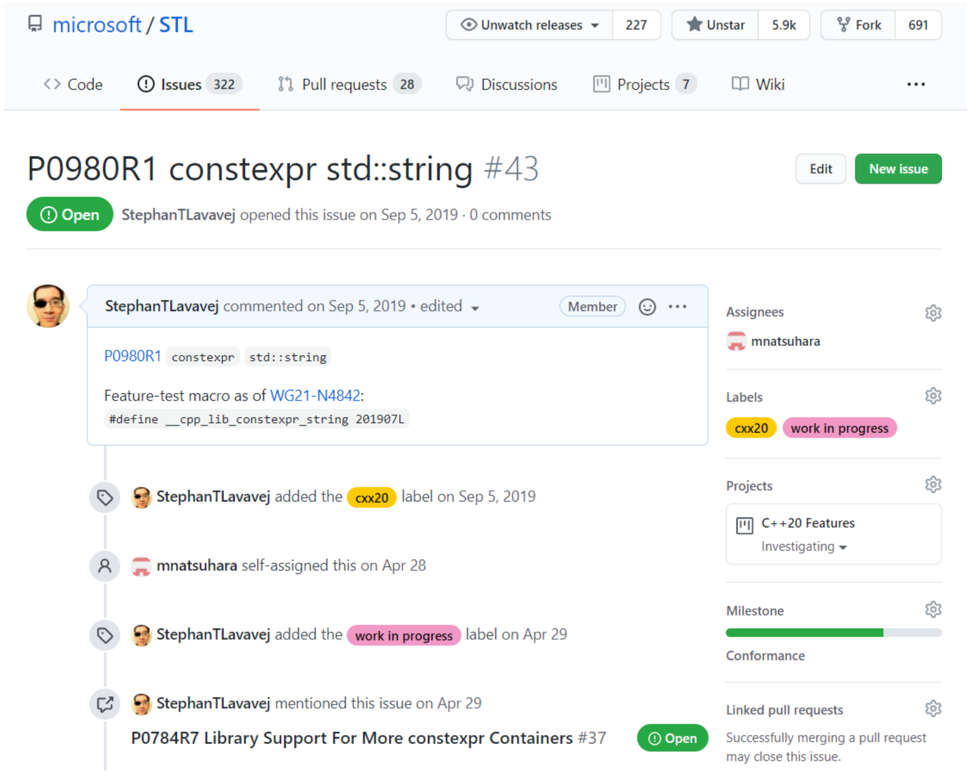
# Code: github.com/microsoft/STL

- Product: `stl/inc`, `stl/src`
- Build system: CMake/Ninja
  - `vcpkg` submodule acquires Boost.Math for Special Math
- `tests/std`, `tests/libcxx`, `tests/tr1` (legacy)
  - `llvm-project` submodule for libc++'s test suite
  - Uses Python and lit (LLVM Integrated Tester)
- Linear history, squashed commits
- Feature branches are uncommon
  - `feature/spaceship`, `feature/format`

## Issues

- `cxx20` issues track all remaining C++20 features
  - GH-39 P0896R4 `<ranges>`
- `LWG` issues track bugfixes in the Standard itself
  - GH-333 LWG-3070 `path::lexically_relative`
- **`bug`**, **`performance`**, **`throughput`**, **`enhancement`**
  - Porting from Developer Community, Azure Boards, todos
  - GH-713 `boyer_moore_searcher`
- **`vNext`** issues affect binary compatibility
  - Will start vNext after finishing C++20

## cxx20 Issues


microsoft / STL

Unwatch releases 227 · Unstar 5.9k · Fork 691

Code · Issues 322 · Pull requests 28 · Discussions · Projects 7 · Wiki

P0980R1 constexpr std::string #43

Edit · New issue

Open · StephanTLavavej opened this issue on Sep 5, 2019 · 0 comments

StephanTLavavej commented on Sep 5, 2019 • edited · Member

P0980R1 `constexpr` `std::string`

Feature-test macro as of WG21-N4842:
`#define __cpp_lib_constexpr_string 201907L`

StephanTLavavej added the cxx20 label on Sep 5, 2019

mnatsuhara self-assigned this on Apr 28

StephanTLavavej added the work in progress label on Apr 29

StephanTLavavej mentioned this issue on Apr 29

P0784R7 Library Support For More constexpr Containers #37 · Open

Assignees
mnatsuhara

Labels
cxx20 · work in progress

Projects
C++20 Features
Investigating

Milestone
Conformance

Linked pull requests
Successfully merging a pull request may close this issue.

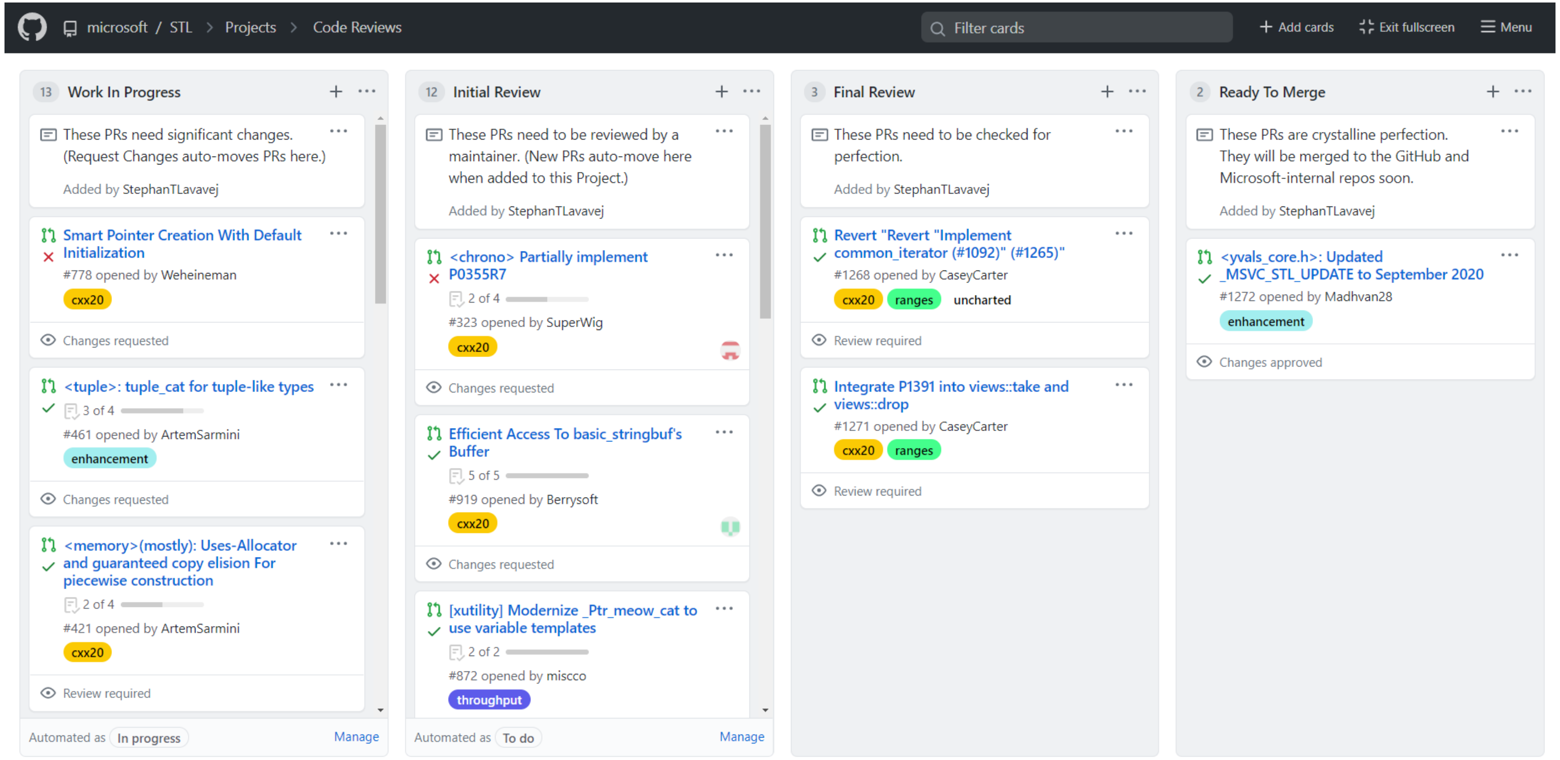
# Pull Requests

- GH-724 Fix `boyer_moore_searcher`
  - Fixed a 43-year-old bug by implementing a 40-year-old fix
  - Explained bug, fix, ABI, testing; others provided more info
- GH-142 P0122R7 `<span>`
  - 691 comments!
- GH-1173 Fix `ostr << flt` precision
  - Fixed long-standing bugs while preserving ABI
- Every PR is extensively reviewed by 2 maintainers
  - Keeps code at production quality: always ready to ship
  - Helps the team understand the code and related features

# Code Review Philosophy

- The Standard Library has:
  - Unusually well-specified preconditions and postconditions
  - Unusually strict correctness and performance requirements
  - Indefinite lifespan: old codebase, yet must keep evolving
- Careful code review avoids regressions, new bugs
  - Especially important for binary compatibility
- Codebase consistency is important, especially now
  - Ideally, looks like the work of a single author
  - Consistent code -> consistent behavior, fewer unique bugs
  - Makes inconsistent code stand out as unusual or incorrect

# Code Reviews Project



microsoft / STL > Projects > Code Reviews

Filter cards | Add cards | Exit fullscreen | Menu

**13 Work In Progress**

These PRs need significant changes. (Request Changes auto-moves PRs here.)
Added by StephanTLavavej

Smart Pointer Creation With Default Initialization
#778 opened by Weheineman
cxx20
Changes requested

<tuple>: tuple_cat for tuple-like types
3 of 4
#461 opened by ArtemSarmini
enhancement
Changes requested

<memory>(mostly): Uses-Allocator and guaranteed copy elision For piecewise construction
2 of 4
#421 opened by ArtemSarmini
cxx20
Review required

Automated as In progress | Manage

**12 Initial Review**

These PRs need to be reviewed by a maintainer. (New PRs auto-move here when added to this Project.)
Added by StephanTLavavej

<chrono> Partially implement P0355R7
2 of 4
#323 opened by SuperWig
cxx20
Changes requested

Efficient Access To basic_stringbuf's Buffer
5 of 5
#919 opened by Berrysoft
cxx20
Changes requested

[xutility] Modernize _Ptr_meow_cat to use variable templates
2 of 2
#872 opened by miscco
throughput

Automated as To do | Manage

**3 Final Review**

These PRs need to be checked for perfection.
Added by StephanTLavavej

Revert "Revert "Implement common_iterator (#1092)" (#1265)"
#1268 opened by CaseyCarter
cxx20 ranges uncharted
Review required

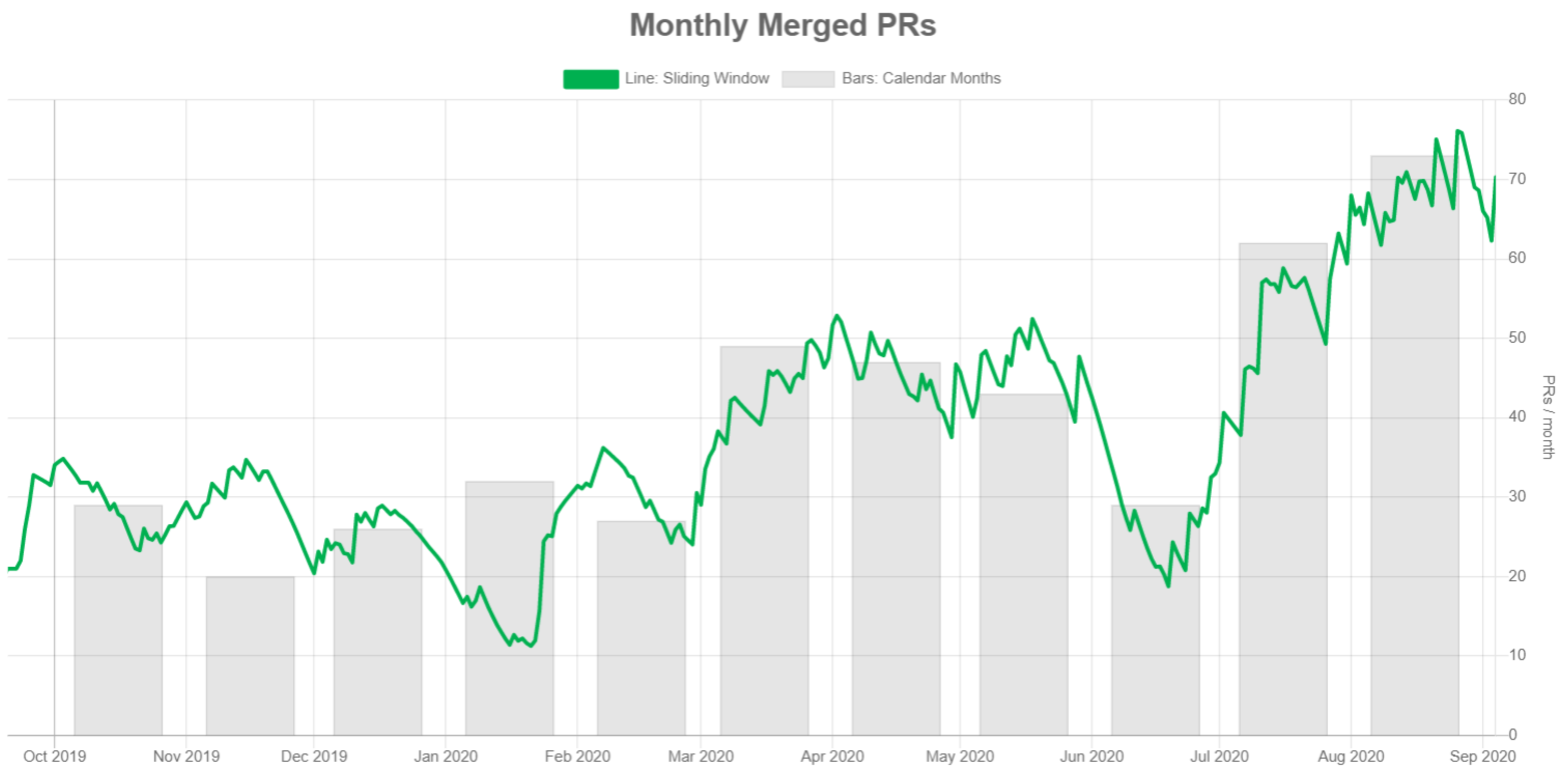
Integrate P1391 into views::take and views::drop
#1271 opened by CaseyCarter
cxx20 ranges
Review required

**2 Ready To Merge**

These PRs are crystalline perfection. They will be merged to the GitHub and Microsoft-internal repos soon.
Added by StephanTLavavej

<yvals_core.h>: Updated _MSVC_STL_UPDATE to September 2020
#1272 opened by Madhvan28
enhancement
Changes approved

# Status Chart: Monthly Merged PRs



## Year 1 ends.
## Year 2 begins!

## More Info

- Links
  - Repository: github.com/microsoft/STL
  - Changelog: github.com/microsoft/STL/wiki/Changelog
  - Status Chart: microsoft.github.io/STL/
  - C++20: wg21.link/n4861
- Questions
  - GitHub Discussions tab
  - Discord server (see README)

## Bonus Slides

# Compile-Time/Run-Time Hybrid

```
template <class _Rx, class _Ty>
_NODISCARD constexpr bool in_range(const _Ty _Value) noexcept {
    // ... see <utility> lines 721-745 ...
    constexpr auto _Ty_min = _Min_limit<_Ty>();
    constexpr auto _Rx_min = _Min_limit<_Rx>();
    if constexpr (_STD cmp_less(_Ty_min, _Rx_min)) {
        if (_Value < _Ty{_Rx_min}) {
            return false;
        }
    }
    // ... similarly for _Value > _Ty{_Rx_max} ...
    return true;
}
```

## Associative Erasure

- `std::erase_if()` is linear time
  - Inspects the entire `value_type`
- Associative containers have member `.erase(key)`
  - Inspects only the key, using the container's predicate
  - Ordered: "Logarithmic" time, O(K + log N)
  - Unordered: "Constant" time, average O(K), worst O(N)
- Only `std::erase_if()` is provided for associative
  - Avoids potential confusion

# Continuous Integration

- Scripts prepare Azure Virtual Machine Scale Sets
  - Currently up to 12 VMs, each with 16 cores
  - VMs install VS (with Clang, CMake, Ninja), Python, CUDA
- We enforce `clang-format` for product/test code
  - Saves an incredible amount of time
  - Our `parallelize` tool runs it quickly, only on C++ files
  - Failed checks display the edits that `clang-format` wants
- Building all architectures is very fast
- Testing x86/x64 takes about an hour
  - Many tests, multiplied by many compiler configurations

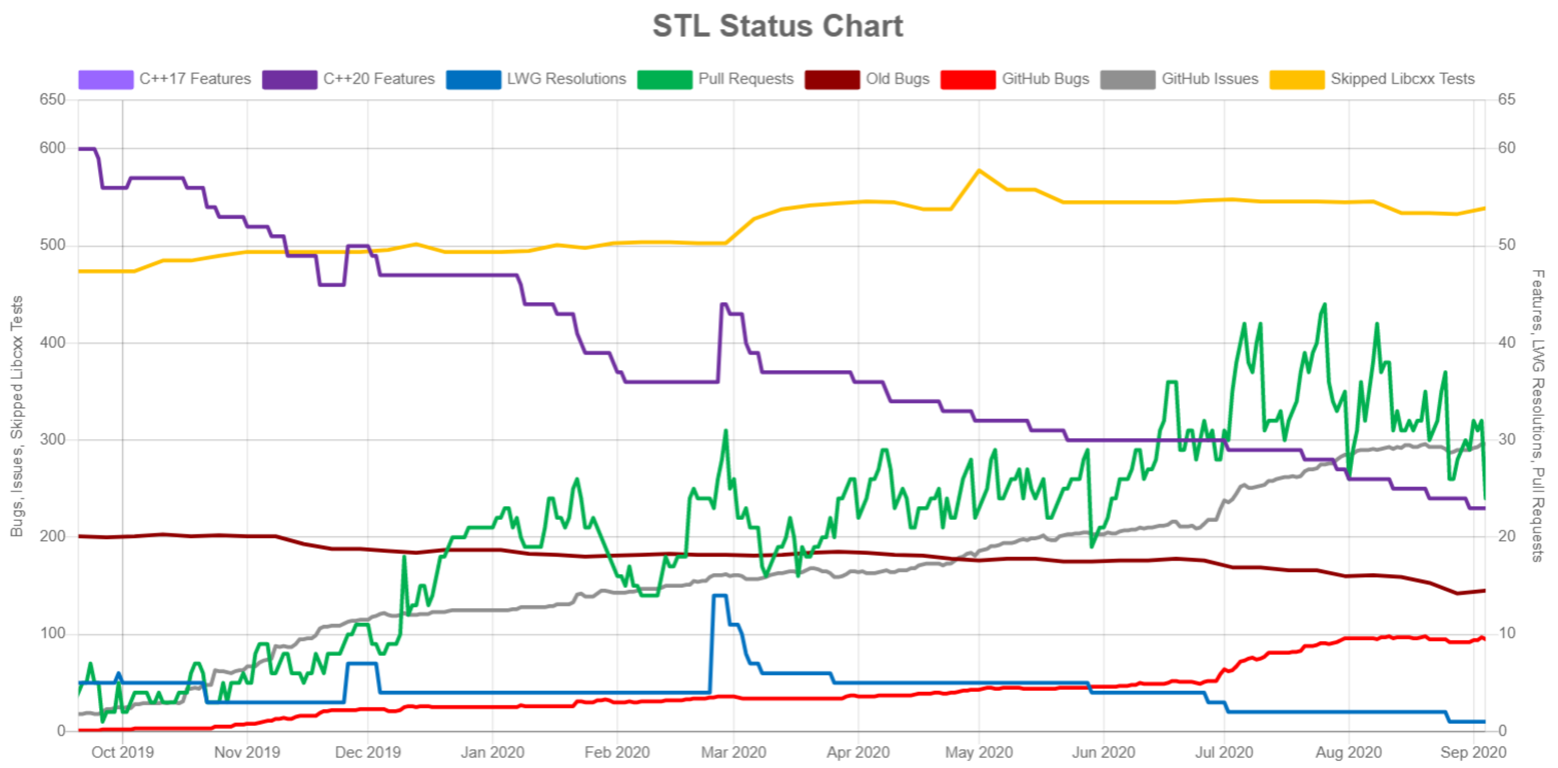
## Projects

- Code Reviews (primary dashboard!)
  - Work In Progress -> Initial Review -> Final Review -> Ready To Merge -> Done
- GitHub Migration
- C++20 Features
  - Available -> Investigating -> Reviewing PR -> Done
- Maintainer Priorities
- Large features: Chrono, Format, Spaceship

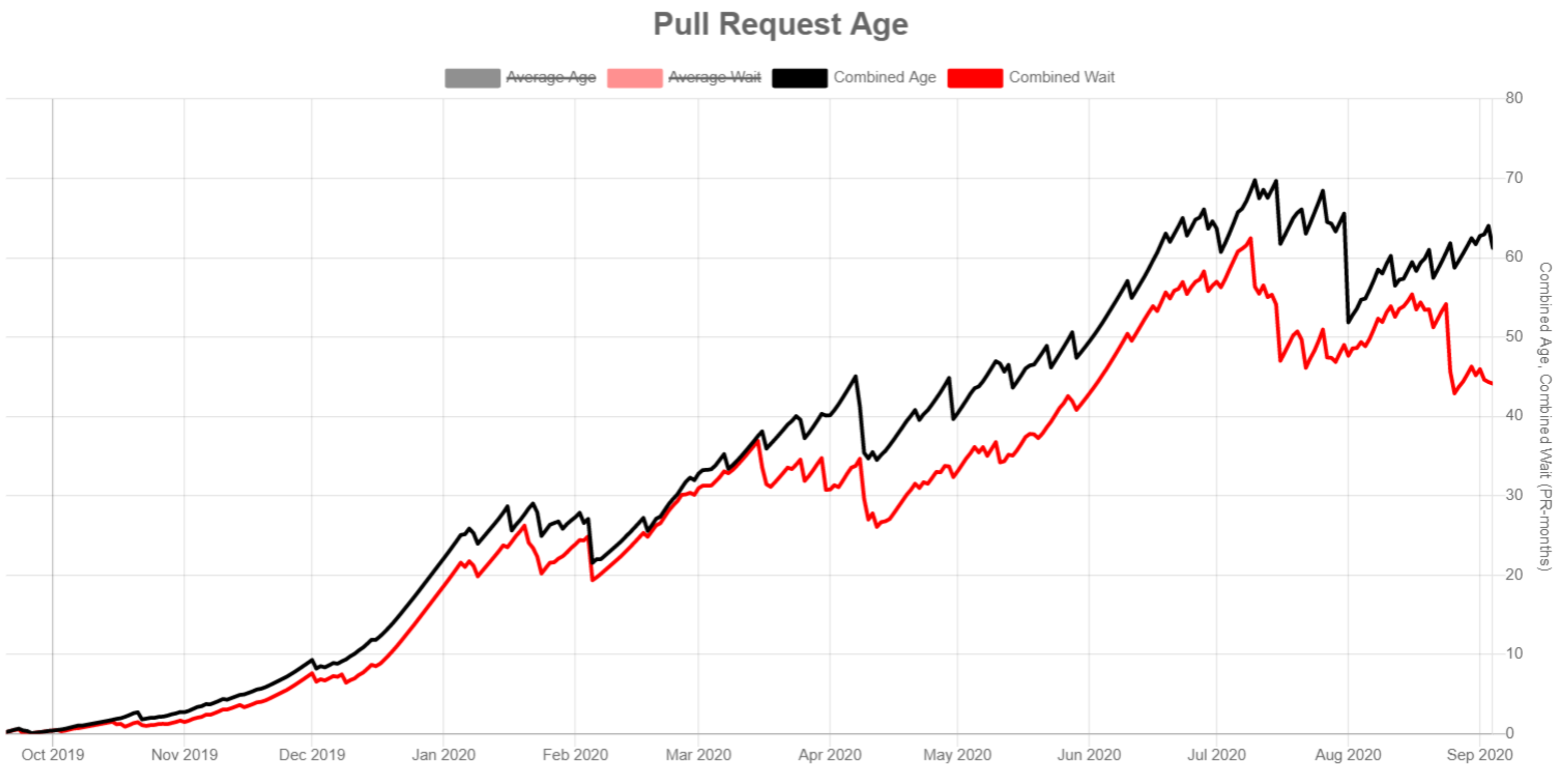
## Wiki

- Changelog
  - Infeasible: Look at commit history between releases
  - Old process: Record commits, write them up later
  - New process: Merge PR, add Changelog entry immediately
- Advice for Reviewing Pull Requests
- MSVC STL Contributors Meeting Notes
- Macro `_MSVC_STL_UPDATE`
  - Updated every month by new contributors
- Custom Autolinks
  - DevCom-724444, LLVM-41915, LWG-3080, WG21-P1209

# Status Chart



# Status Chart: Pull Request Age


Pull Request Age
~~Average Age~~
~~Average Wait~~
Combined Age
Combined Wait
Combined Age, Combined Wait (PR-months)
Oct 2019
Nov 2019
Dec 2019
Jan 2020
Feb 2020
Mar 2020
Apr 2020
May 2020
Jun 2020
Jul 2020
Aug 2020
Sep 2020