Project: ISO JTC1/SC22/WG21: Programming Language C++
Doc No: WG21 **P3725R0**
Date: 2025-06-03
Reply to: Nicolai Josuttis (nico@josuttis.de)
Co-authors:
Audience: SG9, LEWG, LWG
Issues:
Previous:

Filter View Extensions for Input Ranges, Rev 0

Several basic use cases of the current filter view are broken or risky. This paper proposes special features to give programmers of the filter view an easy option to use filter views more intuitive and with less risks.

Motivation

Using filter views is both risky and non-intuitive. Let us look at some applications.

UC1) Possible core dumps

For example (see https://www.godbolt.org/z/qrMYb3G4d):

```
std::vector<std::string> coll1{"Amsterdam", "Berlin", "Cologne", "LA"};

// move long strings in reverse order to another container:
auto large = [](const auto& s) { return s.size() > 5; };
auto sub = coll1 | std::views::filter(large)
                 | std::views::reverse
                 | std::views::as_rvalue
                 | std::ranges::to<std::vector>();
```

This program has undefined behavior and results in a **core dump** due to overwriting memory of other objects.
Note that the core dump depends on the values and predicate used:

- With “Rome” instead of “Berlin” it is “ony” broken (still UB).
- With a predicate “`s.size() > 5`” it is always well defined and works fine

UC2) Healing “broken” elements

For example (https://www.godbolt.org/z/nG3v5vMev):

```
// bring all dead monsters back to live:
auto dead = [] (const auto& m) { return m.isDead(); };
for (auto& m : monsters | std::views::filter(dead)) {
  m.bringBackToLive();  // undefined behavior
}
```

The loop to bring all dead monsters back to live works but is formally undefined behavior. The reason is that it can go wrong when after the filter there are other views like reverse (see the previous example).

UC3) No const iterations supported

For example (see https://www.godbolt.org/z/cjYbsn757, example by Patrice Roy):

```
void constIterate(const auto& coll);

std::vector<std::string> coll3{"Amsterdam", "Berlin", "Cologne", "LA"};

auto large = [](const auto& s) { return s.size() > 5; };
constIterate(coll3 | std::views::filter(large));  // compile-time ERROR
```

All three cases mean that ordinary programmers need good insights of the filter view to understand what is going on and avoid serious mistakes.

This paper provides a simple solutions for these use cases, which is easy to teach and easy to follow. By using std::views::input_filter() to create the filter the broken basic use cases work fine:

```
// bring all dead monsters back to live:
auto dead = [] (const auto& m) { return m.isDead(); };
for (auto& m : monsters | std::views::input_filter(dead)) {
  m.bringBackToLive();  // well defined and works
}

auto large = [](const auto& s) { return s.size() > 5; };
constIterate(coll3 | std::views::input_filter(large));  // OK
```

,

While the broken memory access no longer compiles:

```
std::vector<std::string> coll1{"Amsterdam", "Berlin", "Cologne", "LA"};

// move long strings in reverse order to another container:
auto large = [](const auto& s) { return s.size() > 5; };
auto sub = coll1 | std::views::input_filter(large)
                 | std::views::reverse
                 | std::views::as_rvalue
                 | std::ranges::to<std::vector>();  // compile-time ERROR
```

This compile-time error also occurs in cases previously not broken and working fine. To be able to use filters here, programmers can still use std::views:::filter().

Proposed Changes

The proposed changes are pretty simple:

a) For filter views, add a new **const** member functions **begin()** and **end()** when it operates on an input ranges, which supports a const begin().

b) For filter view iterators, relax the wording of

> Modification of the element a filter_view::*iterator* denotes is permitted, but results in undefined behavior if the resulting value does not satisfy the filter predicate.

to allow all modifications on input iterators:

> Modification of the element a filter_view::*iterator* denotes is permitted, but results in undefined behavior if **the underlying range is a forward_range and** the resulting value does not satisfy the filter predicate.

c) Provide a new range adaptor object **std::views::input_filter** which forces the underlying range internally to be used as input_range using the **to_input** view.

Note that as a result the filter view will have const begin() and end() when operating on an input range. This sounds surprising, because usually pure input ranges read data with begin() (as with ++) and therefore have no const begin().

However, from a semantic perspective, here we have a new special case. Being an input range is provided to make the use of the filter view safer. Technically, begin() is a cheap const operation. We assume that this pattern might also be established for other views in futire.

Proposed Wording

(All against N5008)

In **25.7.8.1 Overview [range.filter.overview]**

Add a new paragraph:

> The name **views::input_filter** denotes a range adaptor object (25.7.2). Given subexpressions E and P, the expression views::filter(E, P) is expression-equivalent to **filter_view(to_input_view(E), P)**.

in **25.7.8.2 Class template filter_view [range.filter.view]**

in the class structure replace:

```
constexpr iterator begin();

constexpr auto end() {
  if constexpr (common_range<V>)
    return iterator {*this, ranges::end(base_)};
  else
    return sentinel {*this};
}
```

by

```
constexpr iterator begin();
constexpr iterator begin() const requires (!forward_range<const V>) &&
                                          indirect_unary_predictate<const Pred&, iterator_t<const V>>;

constexpr auto end() {
  if constexpr (common_range<V>)
    return iterator {*this, ranges::end(base_)};
  else
    return sentinel {*this};
}
constexpr auto end() const requires (!forward_range<const V>)  {
  if constexpr (common_range<const V>)
    return iterator {*this, ranges::end(base_)};
  else
    return sentinel {*this};
}
```

In **24.7.4.2 Class template filter_view [range.filter.view]**

Before the definition of

> **constexpr *iterator* begin();**

add a second member function:

```
constexpr iterator begin() const requires (!forward_range<const V>) &&
                                          indirect_unary_predictate<const Pred&, iterator_t<const V>>
```

using the same effects clause.

In **25.7.8.3 Class filter_view::*iterator* [range.filter.iterator]**

Replace

> Modification of the element a filter_view::*iterator* denotes is permitted, but results in undefined behavior if the resulting value does not satisfy the filter predicate.

by:

> Modification of the element a filter_view::*iterator* denotes is permitted, but results in undefined behavior if **the underlying range is a forward_range and** the resulting value does not satisfy the filter predicate.

Feature Test Macro

Add a new ranges or filter_view feature test macro.

Acknowledgements

Thanks to a lot of people who helped and gave support again and again to finally get this proposal done. Special thanks go to Berry Revzin, Jonathan Müller, Ville Voutilainen, Peter Dimov, Hui Xie, Herb Sutter, Tristan Brindle, which finally took their time to explain the motivation, discuss options, corrected me, and proposed details of the current situation.

Rev0:

First initial version.

References