

constexpr for specialized memory algorithms

Document #: P2283R0
Date: 2021-01-12
Project: Programming Language C++
Audience: Library Evolution Group
Reply-to: Michael Schellenberger Costa
<mschellenbergercosta@gmail.com>

1 Introduction

This paper proposes adding `constexpr` support to the specialized memory algorithms. This is essentially a followup to [\[P0784R7\]](#) which added `constexpr` support for all necessary machinery.

2 Motivation and Scope

These algorithms have been forgotten in the final crunch to get C++20 out. To add insult to injury, they are essential to implementing `constexpr` container support, so every library has to provide its own internal helpers to do the exact same thing during constant evaluation. Just fill the void and add `constexpr` everywhere except the parallel overloads.

But wait, what about `uninitialized_default_construct`? We cannot use `std::construct_at` there! This is correct and there are two possible solutions:

1. Add an overload of `std::construct_at` that takes a `std::default_construct_tag` and does the right thing. This is just bad design. If we go to the trouble of inventing a new name, we should do #2
2. Add a new library function `std::default_construct_at` that does the right thing. This is what is proposed here.

3 Impact on the Standard

This proposal is a pure library extension.

4 Proposed Wording

4.1 Modify 7.7 [expr.const] §6 of [\[N4762\]](#) as follows

For the purposes of determining whether an expression E is a core constant expression, the evaluation of a call to a member function of `std::allocator` as defined in [allocator.members], where T is a literal type, does not disqualify E from being a core constant expression, even if the actual evaluation of such a call would otherwise fail the requirements for a core constant expression. Similarly, the evaluation of a call to `std::destroy_at`, `std::ranges::destroy_at`, [std::default_construct_at](#), [std::ranges::default_construct_at](#), `std::construct_at`, or `std::ranges::construct_at` does not disqualify E from being a core constant expression unless:

for a call to [std::default_construct_at](#), [std::ranges::default_construct_at](#), `std::construct_at` or `std::ranges::construct_at`, the first argument, of type T*, does not point to storage allocated with `std::allocator` or to an object whose lifetime began within the evaluation of E, or the evaluation of the underlying constructor call disqualifies E from being a core constant expression, or

4.2 Modify 20.10.2 [memory.syn] of [N4762] as follows

```
template<class NoThrowForwardIterator>
    constexpr void uninitialized_default_construct(NoThrowForwardIterator first,
                                                    NoThrowForwardIterator last);

template<class ExecutionPolicy, class NoThrowForwardIterator>
    void uninitialized_default_construct(ExecutionPolicy&& exec,          // see [algorithms.parallel.overl
                                        NoThrowForwardIterator first,
                                        NoThrowForwardIterator last);

template<class NoThrowForwardIterator, class Size>
    constexpr NoThrowForwardIterator
        uninitialized_default_construct_n(NoThrowForwardIterator first, Size n);
template<class ExecutionPolicy, class NoThrowForwardIterator, class Size>
    NoThrowForwardIterator
        uninitialized_default_construct_n(ExecutionPolicy&& exec,          // see [algorithms.parallel.overl
                                        NoThrowForwardIterator first, Size n);

namespace ranges {
    template<no-throw-forward-iterator I, no-throw-sentinel-for<I> S>
        requires default_initializable<iter_value_t<I>>
            constexpr I uninitialized_default_construct(I first, S last);
    template<no-throw-forward-range R>
        requires default_initializable<range_value_t<R>>
            constexpr borrowed_iterator_t<R> uninitialized_default_construct(R&& r);

    template<no-throw-forward-iterator I>
        requires default_initializable<iter_value_t<I>>
            constexpr I uninitialized_default_construct_n(I first, iter_difference_t<I> n);
}

template<class NoThrowForwardIterator>
    constexpr void uninitialized_value_construct(NoThrowForwardIterator first,
                                                  NoThrowForwardIterator last);

template<class ExecutionPolicy, class NoThrowForwardIterator>
    void uninitialized_value_construct(ExecutionPolicy&& exec, // see [algorithms.parallel.overloads]
                                       NoThrowForwardIterator first,
                                       NoThrowForwardIterator last);

template<class NoThrowForwardIterator, class Size>
    constexpr NoThrowForwardIterator
        uninitialized_value_construct_n(NoThrowForwardIterator first, Size n);
template<class ExecutionPolicy, class NoThrowForwardIterator, class Size>
    NoThrowForwardIterator
        uninitialized_value_construct_n(ExecutionPolicy&& exec, // see [algorithms.parallel.overloads]
                                       NoThrowForwardIterator first, Size n);

namespace ranges {
    template<no-throw-forward-iterator I, no-throw-sentinel-for<I> S>
        requires default_initializable<iter_value_t<I>>
            constexpr I uninitialized_value_construct(I first, S last);
    template<no-throw-forward-range R>
        requires default_initializable<range_value_t<R>>
            constexpr borrowed_iterator_t<R> uninitialized_value_construct(R&& r);
```

```

template<no-throw-forward-iterator I>
    requires default_initializable<iter_value_t<I>>
    constexpr I uninitialized_value_construct_n(I first, iter_difference_t<I> n);
}

template<class InputIterator, class NoThrowForwardIterator>
    constexpr NoThrowForwardIterator
        uninitialized_copy(InputIterator first, InputIterator last,
                           NoThrowForwardIterator result);
template<class ExecutionPolicy, class InputIterator, class NoThrowForwardIterator>
    NoThrowForwardIterator uninitialized_copy(ExecutionPolicy&& exec, // see [algorithms.parallel.overl
                                             InputIterator first, InputIterator last,
                                             NoThrowForwardIterator result);
template<class InputIterator, class Size, class NoThrowForwardIterator>
    constexpr NoThrowForwardIterator
        uninitialized_copy_n(InputIterator first, Size n, NoThrowForwardIterator result);
template<class ExecutionPolicy, class InputIterator, class Size, class NoThrowForwardIterator>
    NoThrowForwardIterator uninitialized_copy_n(ExecutionPolicy&& exec, // see [algorithms.parallel.overl
                                             InputIterator first, Size n,
                                             NoThrowForwardIterator result);

namespace ranges {
    template<class I, class O>
        using uninitialized_copy_result = in_out_result<I, O>;
    template<input_iterator I, sentinel_for<I> S1,
             no-throw-forward-iterator O, no-throw-sentinel-for<O> S2>
        requires constructible_from<iter_value_t<O>, iter_reference_t<I>>
        constexpr uninitialized_copy_result<I, O>
            uninitialized_copy(I ifirst, S1 ilast, O ofirst, S2 olast);
    template<input_range IR, no-throw-forward-range OR>
        requires constructible_from<range_value_t<OR>, range_reference_t<IR>>
        constexpr uninitialized_copy_result<borrowed_iterator_t<IR>, borrowed_iterator_t<OR>>
            uninitialized_copy(IR&& in_range, OR&& out_range);

    template<class I, class O>
        using uninitialized_copy_n_result = in_out_result<I, O>;
    template<input_iterator I, no-throw-forward-iterator O, no-throw-sentinel-for<O> S>
        requires constructible_from<iter_value_t<O>, iter_reference_t<I>>
        constexpr uninitialized_copy_n_result<I, O>
            uninitialized_copy_n(I ifirst, iter_difference_t<I> n, O ofirst, S olast);
}

template<class InputIterator, class NoThrowForwardIterator>
    constexpr NoThrowForwardIterator
        uninitialized_move(InputIterator first, InputIterator last,
                           NoThrowForwardIterator result);
template<class ExecutionPolicy, class InputIterator, class NoThrowForwardIterator>
    NoThrowForwardIterator uninitialized_move(ExecutionPolicy&& exec, // see [algorithms.parallel.overl
                                             InputIterator first, InputIterator last,
                                             NoThrowForwardIterator result);
template<class InputIterator, class Size, class NoThrowForwardIterator>
    constexpr pair<InputIterator, NoThrowForwardIterator>
        uninitialized_move_n(InputIterator first, Size n, NoThrowForwardIterator result);
template<class ExecutionPolicy, class InputIterator, class Size, class NoThrowForwardIterator>

```

```

pair<InputIterator, NoThrowForwardIterator>
    uninitialized_move_n(ExecutionPolicy&& exec,           // see [algorithms.parallel.overloads]
                        InputIterator first, Size n, NoThrowForwardIterator result);

namespace ranges {
    template<class I, class O>
        using uninitialized_move_result = in_out_result<I, O>;
    template<input_iterator I, sentinel_for<I> S1,
            no-throw-forward-iterator O, no-throw-sentinel-for<O> S2>
        requires constructible_from<iter_value_t<O>, iter_rvalue_reference_t<I>>
        constexpr uninitialized_move_result<I, O>
            uninitialized_move(I ifirst, S1 ilast, O ofirst, S2 olast);
    template<input_range IR, no-throw-forward-range OR>
        requires constructible_from<range_value_t<OR>, range_rvalue_reference_t<IR>>
        constexpr uninitialized_move_result<borrowed_iterator_t<IR>, borrowed_iterator_t<OR>>
            uninitialized_move(IR&& in_range, OR&& out_range);

    template<class I, class O>
        using uninitialized_move_n_result = in_out_result<I, O>;
    template<input_iterator I,
            no-throw-forward-iterator O, no-throw-sentinel-for<O> S>
        requires constructible_from<iter_value_t<O>, iter_rvalue_reference_t<I>>
        constexpr uninitialized_move_n_result<I, O>
            uninitialized_move_n(I ifirst, iter_difference_t<I> n, O ofirst, S olast);
}

template<class NoThrowForwardIterator, class T>
    constexpr void uninitialized_fill(NoThrowForwardIterator first,
                                    NoThrowForwardIterator last, const T& x);
template<class ExecutionPolicy, class NoThrowForwardIterator, class T>
    void uninitialized_fill(ExecutionPolicy&& exec,           // see [algorithms.parallel.overloads]
                          NoThrowForwardIterator first, NoThrowForwardIterator last,
                          const T& x);
template<class NoThrowForwardIterator, class Size, class T>
    constexpr NoThrowForwardIterator
        uninitialized_fill_n(NoThrowForwardIterator first, Size n, const T& x);
template<class ExecutionPolicy, class NoThrowForwardIterator, class Size, class T>
    NoThrowForwardIterator
        uninitialized_fill_n(ExecutionPolicy&& exec,           // see [algorithms.parallel.overloads]
                          NoThrowForwardIterator first, Size n, const T& x);

namespace ranges {
    template<no-throw-forward-iterator I, no-throw-sentinel-for<I> S, class T>
        requires constructible_from<iter_value_t<I>, const T&>
        constexpr I uninitialized_fill(I first, S last, const T& x);
    template<no-throw-forward-range R, class T>
        requires constructible_from<range_value_t<R>, const T&>
        constexpr borrowed_iterator_t<R> uninitialized_fill(R&& r, const T& x);

    template<no-throw-forward-iterator I, class T>
        requires constructible_from<iter_value_t<I>, const T&>
        constexpr I uninitialized_fill_n(I first, iter_difference_t<I> n, const T& x);
}

```

```
// [specialized.construct], construct_at
template<class T, class... Args>
constexpr T* construct_at(T* location, Args&&... args);

namespace ranges {
    template<class T, class... Args>
        constexpr T* construct_at(T* location, Args&&... args);
}
```

```
// [specialized.default_construct], default_construct_at
template<class T>
constexpr T* default_construct_at(T* location);

namespace ranges {
    template<class T>
        constexpr T* default_construct_at(T* location);
}
```

4.3 Modify 25.11.3 [uninitialized.construct.default] of [N4762] as follows

```
template<class NoThrowForwardIterator>
- void uninitialized_default_construct(NoThrowForwardIterator first, NoThrowForwardIterator last);
+ constexpr void uninitialized_default_construct(NoThrowForwardIterator first,
+                                               NoThrowForwardIterator last);

Effects: Equivalent to:
    for (; first != last; ++first)
- ::new (voidify(*first)) typename iterator_traits<NoThrowForwardIterator>::value_type;
+ default_construct_at(to_address(first));

namespace ranges {
    template<no-throw-forward-iterator I, no-throw-sentinel-for<I> S>
        requires default_initializable<iter_value_t<I>>
- I uninitialized_default_construct(I first, S last);
+ constexpr I uninitialized_default_construct(I first, S last);
    template<no-throw-forward-range R>
        requires default_initializable<range_value_t<R>>
- borrowed_iterator_t<R> uninitialized_default_construct(R&& r);
+ constexpr borrowed_iterator_t<R> uninitialized_default_construct(R&& r);
}

Effects: Equivalent to:
    for (; first != last; ++first)
- ::new (voidify(*first)) remove_reference_t<iter_reference_t<I>>;
+ default_construct_at(to_address(first));
    return first;

template<class NoThrowForwardIterator, class Size>
- NoThrowForwardIterator uninitialized_default_construct_n(NoThrowForwardIterator first, Size n);
+ constexpr NoThrowForwardIterator
+ uninitialized_default_construct_n(NoThrowForwardIterator first, Size n);

Effects: Equivalent to:
```

```

    for (; n > 0; (void)++first, --n)
-   ::new (voidify(*first)) typename iterator_traits<NoThrowForwardIterator>::value_type;
+   default_construct_at(to_address(first));
    return first;

namespace ranges {
    template<no-throw-forward-iterator I>
        requires default_initializable<iter_value_t<I>>
-   I uninitialized_default_construct_n(I first, iter_difference_t<I> n);
+   constexpr I uninitialized_default_construct_n(I first, iter_difference_t<I> n);
}

Effects: Equivalent to:
    return uninitialized_default_construct(counted_iterator(first, n),
                                         default_sentinel).base();

```

4.4 Modify 25.11.4 [uninitialized.construct.value] of [N4762] as follows

```

template<class NoThrowForwardIterator>
-   void uninitialized_value_construct(NoThrowForwardIterator first, NoThrowForwardIterator last);
+   constexpr void uninitialized_value_construct(NoThrowForwardIterator first,
+                                               NoThrowForwardIterator last);

Effects: Equivalent to:
    for (; first != last; ++first)
-   ::new (voidify(*first)) typename iterator_traits<NoThrowForwardIterator>::value_type();
+   construct_at(to_address(first));

namespace ranges {
    template<no-throw-forward-iterator I, no-throw-sentinel-for<I> S>
        requires value_initializable<iter_value_t<I>>
-   I uninitialized_value_construct(I first, S last);
+   constexpr I uninitialized_value_construct(I first, S last);
    template<no-throw-forward-range R>
        requires value_initializable<range_value_t<R>>
-   borrowed_iterator_t<R> uninitialized_value_construct(R&& r);
+   constexpr borrowed_iterator_t<R> uninitialized_value_construct(R&& r);
}

Effects: Equivalent to:
    for (; first != last; ++first)
-   ::new (voidify(*first)) remove_reference_t<iter_reference_t<I>>();
+   construct_at(to_address(first));
    return first;

template<class NoThrowForwardIterator, class Size>
-   NoThrowForwardIterator uninitialized_value_construct_n(NoThrowForwardIterator first, Size n);
+   constexpr NoThrowForwardIterator
+   uninitialized_value_construct_n(NoThrowForwardIterator first, Size n);

Effects: Equivalent to:
    for (; n > 0; (void)++first, --n)
-   ::new (voidify(*first)) typename iterator_traits<NoThrowForwardIterator>::value_type();

```

```

+   construct_at(to_address(first));
return first;

namespace ranges {
    template<no-throw-forward-iterator I>
        requires value_initializable<iter_value_t<I>>
-       I uninitialized_value_construct_n(I first, iter_difference_t<I> n);
+       constexpr I uninitialized_value_construct_n(I first, iter_difference_t<I> n);
}

Effects: Equivalent to:
    return uninitialized_value_construct(counted_iterator(first, n),
                                       value_sentinel).base();

```

4.5 Modify 25.11.5 [uninitialized.copy] of [N4762] as follows

```

template<class InputIterator, class NoThrowForwardIterator>
-   NoThrowForwardIterator uninitialized_copy(InputIterator first, InputIterator last,
-                                           NoThrowForwardIterator result);
+   constexpr NoThrowForwardIterator
+       uninitialized_copy(InputIterator first, InputIterator last,
+                           NoThrowForwardIterator result);

Preconditions:
    result + [0, (last - first)) does not overlap with [first, last).

Effects: Equivalent to:
    for (; first != last; ++result, (void) ++first)
-       ::new (voidify(*result))
-       typename iterator_traits<NoThrowForwardIterator>::value_type(*first);
+       construct_at(to_address(result), *first);

Returns: result.

namespace ranges {
    template<input_iterator I, sentinel_for<I> S1,
            no-throw-forward-iterator O, no-throw-sentinel-for<O> S2>
        requires constructible_from<iter_value_t<O>, iter_reference_t<I>>
-       uninitialized_copy_result<I, O>
+       constexpr uninitialized_copy_result<I, O>
            uninitialized_copy(I ifirst, S1 ilast, O ofirst, S2 olast);
    template<input_range IR, no-throw-forward-range OR>
        requires constructible_from<range_value_t<OR>, range_reference_t<IR>>
-       uninitialized_copy_result<borrowed_iterator_t<IR>, borrowed_iterator_t<OR>>
+       constexpr uninitialized_copy_result<borrowed_iterator_t<IR>, borrowed_iterator_t<OR>>
            uninitialized_copy(IR&& in_range, OR&& out_range);
}

Preconditions:
    [ofirst, olast) does not overlap with [ifirst, ilast).

Effects: Equivalent to:
    for (; ifirst != ilast && ofirst != olast; ++ofirst, (void)++ifirst)

```

```

-      ::new (voidify(*ofirst)) remove_reference_t<iter_reference_t<O>>(*ifirst);
+      construct_at(to_address(ofirst), *ifirst);
      return {std::move(ifirst), ofirst};

template<class InputIterator, class Size, class NoThrowForwardIterator>
-      NoThrowForwardIterator uninitialized_copy_n(InputIterator first, Size n,
-                                                  NoThrowForwardIterator result);
+      constexpr NoThrowForwardIterator
+      uninitialized_copy_n(InputIterator first, Size n, NoThrowForwardIterator result);

Preconditions:
      result + [0, n) does not overlap with first + [0, n).

Effects: Equivalent to:
      for ( ; n > 0; ++result, (void) ++first, --n)
-      ::new (voidify(*result))
-      typename iterator_traits<NoThrowForwardIterator>::value_type(*first);
+      construct_at(to_address(result), *first);

Returns: result.

namespace ranges {
    template<input_iterator I, no-throw-forward-iterator O, no-throw-sentinel-for<O> S>
        requires constructible_from<iter_value_t<O>, iter_reference_t<I>>
-        uninitialized_copy_n_result<I, O>
+        constexpr uninitialized_copy_n_result<I, O>
            uninitialized_copy_n(I ifirst, iter_difference_t<I> n, O ofirst, S olast);
}

Preconditions:
    [ofirst, olast) does not overlap with ifirst + [0, n).

Effects: Equivalent to:
    auto t = uninitialized_copy(counted_iterator(ifirst, n),
                                default_sentinel, ofirst, olast);
    return {std::move(t.in).base(), t.out};

```

4.6 Modify 25.11.6 [uninitialized.move] of [N4762] as follows

```

template<class InputIterator, class NoThrowForwardIterator>
-      NoThrowForwardIterator uninitialized_move(InputIterator first, InputIterator last,
-                                                  NoThrowForwardIterator result);
+      constexpr NoThrowForwardIterator
+      uninitialized_move(InputIterator first, InputIterator last, NoThrowForwardIterator result);

Preconditions:
      result + [0, (last - first)) does not overlap with [first, last).

Effects: Equivalent to:
      for (; first != last; ++result, (void) ++first)
-      ::new (voidify(*result))
-      typename iterator_traits<NoThrowForwardIterator>::value_type(std::move(*first));
+      construct_at(to_address(result), std::move(*first));

```


Returns: result.

```
namespace ranges {
    template<input_iterator I, sentinel_for<I> S1,
            no-throw-forward-iterator O, no-throw-sentinel-for<O> S2>
        requires constructible_from<iter_value_t<O>, iter_rvalue_reference_t<I>>
-       uninitialized_move_result<I, O>
+       constexpr uninitialized_move_result<I, O>
        uninitialized_move(I ifirst, S1 ilast, O ofirst, S2 olast);
    template<input_range IR, no-throw-forward-range OR>
        requires constructible_from<range_value_t<OR>, range_rvalue_reference_t<IR>>
-       uninitialized_move_result<borrowed_iterator_t<IR>, borrowed_iterator_t<OR>>
+       constexpr uninitialized_move_result<borrowed_iterator_t<IR>, borrowed_iterator_t<OR>>
        uninitialized_move(IR&& in_range, OR&& out_range);
}
```

Preconditions:

[ofirst, olast) does not overlap with [ifirst, ilast).

Effects: Equivalent to:

for (; ifirst != ilast && ofirst != olast; ++ofirst, (void)++ifirst)

```
-       ::new (voidify(*ofirst))
-       remove_reference_t<iter_reference_t<O>>(ranges::iter_move(ifirst);
+       construct_at(to_address(ofirst), ranges::iter_move(ifirst);
return {std::move(ifirst), ofirst};
```

[Note 1: If an exception is thrown, some objects in the range [first, last) are left in a valid, but un

```
template<class InputIterator, class Size, class NoThrowForwardIterator>
```

```
-   NoThrowForwardIterator uninitialized_move_n(InputIterator first, Size n,
-                                               NoThrowForwardIterator result);
+   constexpr NoThrowForwardIterator
+   uninitialized_move_n(InputIterator first, Size n, NoThrowForwardIterator result);
```

Preconditions:

result + [0, n) does not overlap with first + [0, n).

Effects: Equivalent to:

for (; n > 0; ++result, (void)++first, --n)

```
-       ::new (voidify(*result))
-       typename iterator_traits<NoThrowForwardIterator>::value_type(std::move(*first));
+       construct_at(to_address(result), std::move(*first));
```

Returns: result.

```
namespace ranges {
    template<input_iterator I, no-throw-forward-iterator O, no-throw-sentinel-for<O> S>
        requires constructible_from<iter_value_t<O>, iter_rvalue_reference_t<I>>
-       uninitialized_move_n_result<I, O>
+       constexpr uninitialized_move_n_result<I, O>
        uninitialized_move_n(I ifirst, iter_difference_t<I> n, O ofirst, S olast);
}
```

Preconditions:

[ofirst, olast) does not overlap with ifirst + [0, n).

Effects: Equivalent to:

```
auto t = uninitialized_move(counted_iterator(ifirst, n),
                             default_sentinel, ofirst, olast);
return {std::move(t.in).base(), t.out};
```

[Note 2: If an exception is thrown, some objects in the range first + [0, n) are left in a valid but un

4.7 Modify 25.11.7 [uninitialized.fill] of [N4762] as follows

```
template<class NoThrowForwardIterator, class T>
- void uninitialized_fill(NoThrowForwardIterator first, NoThrowForwardIterator last, const T& x);
+ constexpr void uninitialized_fill(NoThrowForwardIterator first,
+                                   NoThrowForwardIterator last, const T& x);
Effects: Equivalent to:
    for (; first != last; ++first)
-        ::new (voidify(*first))
-            typename iterator_traits<NoThrowForwardIterator>::value_type(x);
+        construct_at(to_address(first), x);

namespace ranges {
    template<no-throw-forward-iterator I, no-throw-sentinel-for<I> S, class T>
        requires constructible_from<iter_value_t<I>, const T&>
-        I uninitialized_fill(I first, S last, const T& x);
+        constexpr I uninitialized_fill(I first, S last, const T& x);
    template<no-throw-forward-range R, class T>
        requires constructible_from<range_value_t<R>, const T&>
-        borrowed_iterator_t<R> uninitialized_fill(R&& r, const T& x);
+        constexpr borrowed_iterator_t<R> uninitialized_fill(R&& r, const T& x);
}

Effects: Equivalent to:
    for (; first != last; ++first)
-        ::new (voidify(*first)) remove_reference_t<iter_reference_t<I>>(x);
+        construct_at(to_address(first), x);
    return first;

template<class NoThrowForwardIterator, class Size, class T>
- NoThrowForwardIterator uninitialized_fill_n(NoThrowForwardIterator first, Size n, const T& x);
+ constexpr NoThrowForwardIterator uninitialized_fill_n(NoThrowForwardIterator first,
+                                                         Size n, const T& x);

Effects: Equivalent to:
    for (; n--; ++first)
-        ::new (voidify(*first))
-            typename iterator_traits<NoThrowForwardIterator>::value_type(x);
+        construct_at(to_address(first), x);
    return first;

namespace ranges {
    template<no-throw-forward-iterator I, class T>
        requires constructible_from<iter_value_t<I>, const T&>
```

```

-   I uninitialized_fill_n(I first, iter_difference_t<I> n, const T& x);
+   constexpr I uninitialized_fill_n(I first, iter_difference_t<I> n, const T& x);
}

```

Effects: Equivalent to:

```
return uninitialized_fill(counted_iterator(first, n), default_sentinel, x).base();
```

4.8 Add 25.11.8 [special.default_construct] to [N4762]

```

template<class T>
constexpr T* default_construct_at(T* location);

namespace ranges {
    template<class T>
    constexpr T* default_construct_at(T* location);
}

```

Constraints:

The expression `::new (declval<void*>()) T` is well-formed when treated as an unevaluated operand.

Effects: Equivalent to:

```
return ::new (voidify(*location)) T;
```

5 Implementation Experience

— [Microsoft STL](#) This has been partially implemented for support of constexpr vector in MSVC STL.

6 References

[N4762] Richard Smith. 2018-07-07. Working Draft, Standard for Programming Language C++.

<https://wg21.link/n4762>

[P0784R7] 2019. More constexpr containers.

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0784r7.html>