Add `std::is_partitioned_until` algorithm

Document #:	P1927R1
Date:	2020-02-20
Project:	Programming Language C++
Audience:	LEWGI

Reply-to: Alexander Zaitsev <zamazan4ik@tut.by, zamazan4ik@gmail.com>

1 Revision history

- R1
  - Add a remark about naming: why `is_*` naming was chosen
  - Change for the parallel version signature: use ForwardIterator instead of InputIterator (for consistency with the existing parallel algorithms and also likely a prerequisite for implementers to be able to implement this)
  - Change return value for the third overload: from `input_iterator` to `I`
  - Change return value for the forth overload: from `iterator_t<R>` to `safe_iterator_t<R>`
- R0
  - Initial draft

2 Motivation

`std::is_partitioned` was added long time ago to the standard library. The algorithm is useful but sometimes we need an infromation about "where a partition is broken". `std::is_partitioned` returns only `bool` and we cannot change an interface of existing function. So we can add additional function `std::is_partitioned_until` which returns an iterator instead of bool. `partition_point` cannot be used here because it works only on partitioned ranges.

3 Naming

`std::is_partitioned_until` as a name was chosen only for being consistent with already defined in the standard library `is_sorted_until` function. If anyone can propose better name - it can be discussed again (however I didn't find better name for such function). But from my point of view we shall be consistent here with existent functionality in the standard library.

4 Proposed wording

Add to [**alg.partitions**] **25.7.4**:

[...]

```
template<class InputIterator, class Predicate>
  constexpr InputIterator is_partitioned_until(InputIterator first, InputIterator last,
                                               Predicate pred);
template<class ExecutionPolicy, class ForwardIterator, class Predicate>
  ForwardIterator is_partitioned_until(ExecutionPolicy&& exec, ForwardIterator first,
                                       ForwardIterator last, Predicate pred);

template<input_iterator I, sentinel_for<I> S, class Proj = identity,
         indirect_unary_predicate<projected<I, Proj>> Pred>
  constexpr I ranges::is_partitioned_until(I first, S last, Pred pred,
                                           Proj proj = {});
template<input_range R, class Proj = identity,
         indirect_unary_predicate<projected<iterator_t<R>, Proj>> Pred>
  constexpr borrowed_iterator_t<R> ranges::is_partitioned_until(R&& r, Pred pred,
                                                                Proj proj = {});
```

Let proj be identity for the overloads with no parameter named proj.

Returns: The last iterator `it` in the sequence `[first, last)` for which the `is_partitioned(first, it)` is true.

Complexity: Linear. At most `last - first` applications of `pred` and `proj`.

[...]

5 Examples

Given the container c containing `0,1,2,3,14,15`, then

```
bool isOdd ( int i ) { return i % 2 == 1; }
bool lessThan10 ( int i ) { return i < 10; }

is_partitioned_until ( c, isOdd ) // iterator to '1'
is_partitioned_until ( c, lessThan10 ) // end
is_partitioned_until ( c.begin (), c.end (), lessThan10 ) // end
is_partitioned_until ( c.begin (), c.begin () + 3, lessThan10 ) // end
is_partitioned_until ( c.end (), c.end (), isOdd ) // end, because of empty range
```

6 Possible implementation

Also possible implementation can be found in Boost.Algorithm: GitHub. Documentation can be found here: Boost. Available in Boost.Algorithm since Boost 1.65.