| | |
|---|---|
| **Document Number:** | P1474R1 |
| **Date:** | 2019-07-19 |
| **Audience:** | Library Working Group |
| **Author:** | Casey Carter |
| **Reply to:** | casey@carter.net |

# Helpful pointers for `ContiguousIterator`

## 1 Abstract

The support for contiguous iterators in the working draft is missing a useful feature: a mechanism to convert a contiguous iterator into a pointer that denotes the same object. This paper proposes that `std::to_address` be that mechanism.

Table 1 — Tony Table

| Before | After |
|---|---|
| `extern "C" int some_c_api(T* ptr, size_t size);`<br>`extern "C" int other_c_api(T* first, T* last);`<br><br>`template<ContiguousIterator I>`<br>`int try_useful_things(I i, size_t n) {`<br>  // *Expects: [i, n) is a valid range*<br>  ~~if (n == 0) {~~<br>    ~~// Oops - can't dereference~~<br>    ~~// past-the-end iterator~~<br>    ~~throw something;~~<br>  ~~}~~<br>  return some_c_api(~~addressof(*i)~~, n);<br>}<br><br>`template<ContiguousIterator I>`<br>`int try_useful_things(I i, I j) {`<br>  // *Expects: [i, j) is a valid range*<br>  ~~if (i == j) {~~<br>    ~~// Oops - can't dereference~~<br>    ~~// past-the-end iterator~~<br>    ~~throw something;~~<br>  }<br>  return other_c_api(~~addressof(*i)~~,<br>                     ~~addressof(*i)~~ + (j - i));<br>} | `extern "C" int some_c_api(T* ptr, size_t size);`<br>`extern "C" int other_c_api(T* first, T* last);`<br><br>`template<ContiguousIterator I>`<br>`int try_useful_things(I i, size_t n) {`<br>  // *Expects: [i, n) is a valid range*<br>  `return some_c_api(to_address(i), n);`<br>}<br><br>`template<ContiguousIterator I>`<br>`int try_useful_things(I i, I j) {`<br>  // *Expects: [i, j) is a valid range*<br>  `return other_c_api(to_address(i),`<br>                     `to_address(j));`<br>} |

### 1.1 Revision History

#### 1.1.1 Revision 1

- Update Tony Table: C APIs can't be overloaded, and add a bit of markup to make the differences stand out.
- Correct bad pointer arithmetic in the description of the address of a past-the-end iterator whose predecessor is dereferenceable.
- Remove bad `;` after expression in a *compound-requirement* in the definition of `ContiguousIterator`.
- Remove `operator->` requirement (which was not a core part of the proposal) due to LWG concerns.

### 1.1.2 Revision 0

- Initial revision.

# 2 Problem description

P0944R0 “Contiguous ranges” [1] proposed support for contiguous ranges and iterators, which was merged into P0896R4 “The One Ranges Proposal” [?] and then merged into the Working Draft. Neither P0944R0 nor P0896R4 proposed a means of obtaining a pointer to the element denoted by an arbitrary `ContiguousIterator`. At the time, the author was under the impression that such a mechanism had been a “third rail” for past contiguous iterator proposals [3], and that requiring such a mechanism would make it impossible to require the iterators of the Standard Library containers to model `ContiguousIterator`. Those implementability concerns have since been rectified.

Note that obtaining a pointer value from a dereferenceable `ContiguousIterator` is trivial: `std::addressof(*i)` returns such a pointer value for a contiguous iterator `i`. Dereferencing a non-dereferenceable iterator is (unsurprisingly) not well-defined, so this mechanism isn’t suitable for iterators not known to be dereferenceable. Obtaining a pointer value for the potentially non-dereferenceable iterator `j` that is the past-the-end iterator of a range `[i, j)` thus requires a different mechanism that is well-defined for past-the-end iterators. Ideally the mechanism would also be well-defined for dereferenceable iterators so it can be used uniformly.

P0653R2 “Utility to convert a pointer to a raw pointer” [2] added the function `std::to_address` ([pointer.conversion]) to the Standard Library which converts values of so-called “fancy” pointer types and standard smart pointer types to pointer values. In the interest of spelling similar things similarly, it seems a good idea to reuse this facility to convert `ContiguousIterator`s to pointer values. In practice, that means that a type `I` must be a pointer type or

- specialize `pointer_traits<I>` with a member `element_type` or have a nested member `element_type` so instantiation of `pointer_traits<I>` succeeds, and
- Either implement `pointer_traits<I>::to_address` or admit past-the-end (potentially non-dereferenceable) iterator values in `operator->()`.

# 3 Proposal

The basic proposal is to add a requirement to the `ContiguousIterator` concept that the expression `std::to_address(i)` for an lvalue `i` of type `const I` must

- be well-formed and yield a pointer of type `add_pointer_t<iter_reference_t<i>>`,
- be well-defined for both dereferenceable and past-the-end pointer values,
- yield a pointer value equal to `std::addressof(*i)` if `i` is dereferenceable, or `1 + std::addressof(*(i - 1))` if `i - 1` is dereferenceable.

Since dereferenceable `ContiguousIterator`s always denote objects - their reference types are always lvalue references - they can always feasibly implement the `->` operator. `->` is useful in contexts where the value type of the iterator is concrete, so we propose requiring it for all `ContiguousIterator`s. [ *Note:* Recall that the iterator concepts do not generally require `operator->` as do the “old” iterator requirements. — *end note* ]

Now that there’s a mechanism to retrieve a pointer from a potentially non-dereferenceable iterator, we can also cleanup the edge cases in `ranges::data` and `ranges::view_interface::data` which return `nullptr` for an empty `ContiguousRange` rather than unconditionally returning the pointer value that the `begin` iterator denotes.

# 4 Technical specifications

Change [iterator.concept.contiguous] as follows:

```
template<class I>
  concept ContiguousIterator =
    RandomAccessIterator<I> &&
    DerivedFrom<ITER_CONCEPT(I), contiguous_iterator_tag> &&
    is_lvalue_reference_v<iter_reference_t<I>> &&
    Same<iter_value_t<I>, remove_cvref_t<iter_reference_t<I>>>~~;~~ &&
    requires(const I& i) {
      { to_address(i) } -> Same<add_pointer_t<iter_reference_t<I>>>;
    };
```

2 Let `a` and `b` be dereferenceable iterators and `c` a non-dereferenceable iterator of type `I` such that `b` is reachable from `a` and `c` is reachable from `b`, and let `D` be `iter_difference_t<I>`. The type `I` models `ContiguousIterator` only if `addressof(*(a + D(b - a)))` is equal to `addressof(*a) + D(b - a)`.

(2.1) — `to_address(a) == addressof(*a)`,

(2.2) — `to_address(b) == to_address(a) + D(b - a)`, and

(2.3) — `to_address(c) == to_address(a) + D(c - a)`.

Change [range.prim.data] as follows:

1 The name `data` denotes a customization point object ([customization.point.object]). The expression `ranges::data(E)` for some subexpression `E` is expression-equivalent to:

(1.1) — If `E` is an lvalue, ***`decay-copy`***`(E.data())` if it is a valid expression of pointer to object type.

(1.2) — Otherwise, if `ranges::begin(E)` is a valid expression whose type models `ContiguousIterator`, `to_address(ranges::begin(E))`.

```
ranges::begin(E) == ranges::end(E) ? nullptr : addressof(*ranges::begin(E))
```

except that `E` is evaluated only once.

(1.3) — Otherwise, `ranges::data(E)` is ill-formed. [ *Note:* This case can result in substitution failure when `ranges::data(E)` appears in the immediate context of a template instantiation. — *end note* ]

Change [view.interface] as follows:

```
namespace std::ranges {
  template<class D>
    requires is_class_v<D> && Same<D, remove_cv_t<D>>
  class view_interface : public view_base {
    [...]

    constexpr auto data() requires ContiguousIterator<iterator_t<D>> {
      ~~return ranges::empty(derived()) ? nullptr : addressof(*ranges::begin(derived()));~~
      return to_address(ranges::begin(derived()));
    }
    constexpr auto data() const
      requires Range<const D> && ContiguousIterator<iterator_t<const D>> {
        ~~return ranges::empty(derived()) ? nullptr : addressof(*ranges::begin(derived()));~~
        return to_address(ranges::begin(derived()));
      }

    [...]
  };
}
```

# Bibliography

[1] Casey Carter. P0944R0: Contiguous ranges, 02 2018. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0944r0.html.

[2] Glen Joseph Fernandes. P0653R2: Utility to convert a pointer to a raw pointer, 11 2017. https://wg21.link/p0653r2.

[3] Nevin "=)" Liber. N4183: Contiguous iterators: Pointer conversion and type trait, 10 2014. https://wg21.link/n4183.