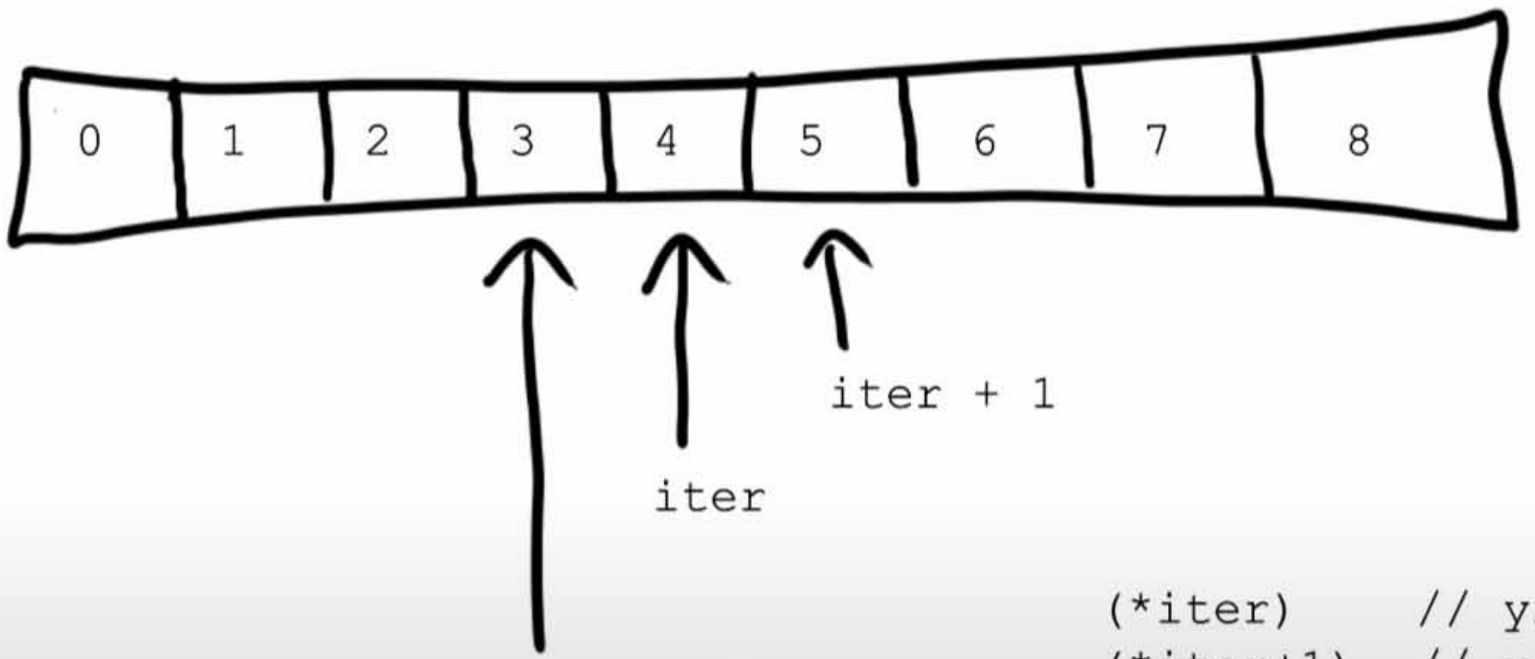
Iterators



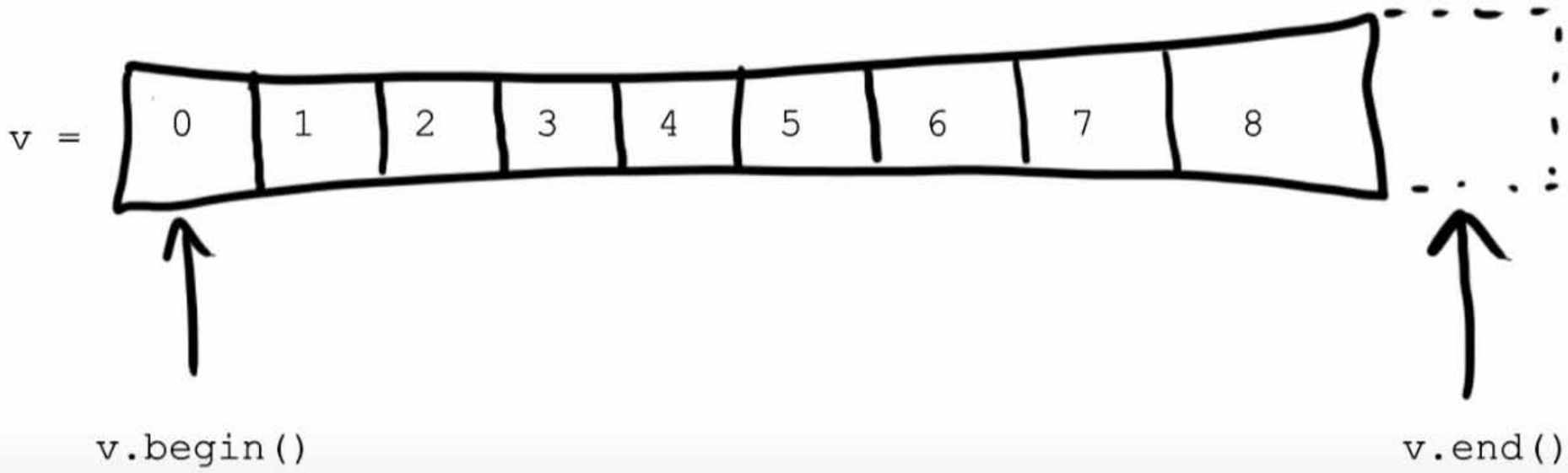

```
(*iter)     // yields 4
(*iter+1)   // yields 5
(*iter-1)   // yields 3
```

<table>
<tr><th colspan="5">Iterator category</th><th>Defined operations</th></tr>
<tr><td rowspan="4">LegacyContiguousIterator</td><td rowspan="4">LegacyRandomAccessIterator</td><td rowspan="3">LegacyBidirectionalIterator</td><td rowspan="2">LegacyForwardIterator</td><td>LegacyInputIterator</td><td>• read
• increment (without multiple passes)</td></tr>
<tr><td rowspan="2"></td><td>• increment (with multiple passes)</td></tr>
<tr><td colspan="2"></td><td>• decrement</td></tr>
<tr><td colspan="3"></td><td>• random access</td></tr>
<tr><td colspan="4"></td><td>• contiguous storage</td></tr>
<tr><td colspan="6">Iterators that fall into one of the above categories and also meet the requirements of LegacyOutputIterator are called mutable iterators.</td></tr>
<tr><td colspan="4">LegacyOutputIterator</td><td>• write
• increment (without multiple passes)</td></tr>
</table>

Note: *LegacyContiguousIterator* category was only formally specified in C++17, but the iterators of `std::vector`, `std::basic_string`, `std::array`, and `std::valarray`, as well as pointers into C arrays are often treated as a separate category in pre-C++17 code.

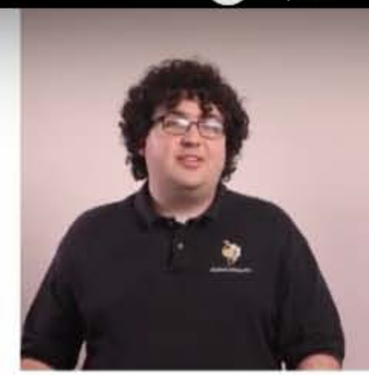
https://en.cppreference.com/w/cpp/iterator

Getting iterators from containers




```
// Get read-only iterators
v.cbegin()
v.cend()
```

```
// Get reverse iterators
v.rbegin()
v.rend()
```



Special Iterator Types

- **Defined in the `<iterator>` header**
- "Inserter Iterators"
  - Writable iterators which add elements to the range when written to.

https://www.youtube.com/watch?v=2olsGf6JIkU

```
std::vector<Card> GenerateDeck() {
    std::vector<Card> deck;
    std::array<int, 13> values;
    std::iota(values.begin(), values.end(), 1);
    std::array<Suit, 4> suits = {Clubs, Diamonds, Hearts, Spades};
    for(const auto suit : suits) {
        std::transform(values.begin(), values.end(), std::back_inserter(deck), [suit](const auto value){
        return Card{suit, value};
        });
    }
    return deck;
}
```

outer_product