Iterators

Victor Eijkhout, Susan Lindsey

Fall 2022

last formatted: March 27, 2023

Begin/end iterator

1. Containers have iterators

Containers such as `vector`, `map` have `begin/end` iterators.

'Pointer' (not in the technical sense) to first and one-beyond-last element.

2. Using iterators

- An iterator is a little like a pointer
  (into anything iteratable)
- *begin* / *end*
- pointer-arithmetic and 'dereferencing':

```
auto element_ptr = my_vector.begin();
element_ptr++;
cout << *element_ptr;
```

- allows operations (erase, insert) on containers:
  erase/insert elements at some location given by an iterator

3. Begin and end iterator

Use independent of looping:

Code:

```
1     vector<int> v{1,3,5,7};
2     auto pointer = v.begin();
3     cout << "we start at "
4          << *pointer << '\n';
5     pointer++;
6     cout << "after increment: "
7          << *pointer << '\n';
8
9     pointer = v.end();
10    cout << "end is not a valid
       element: "
11         << *pointer << '\n';
12    pointer--;
13    cout << "last element: "
14         << *pointer << '\n';
```

Output:

```
we start at 1
after increment: 3
end is not a valid
     element: 0
last element: 7
```

4. (In case you know C)

This is not a C-style pointer dereference,
but rather an overloaded oeprator.

5. Copy range

Copy a begin/end range of one container
to an iterator in another container::

Code:

```
1 vector<int> counts{1,2,3,4};
2 vector<int> copied(5);
3 copy( counts.begin(),counts.end(),
4       copied.begin()+1 );
5 cout << copied[0]
6      << ", " << copied[1]
7      << ".." << copied[4] << '\n';
```

Output:

```
0, 1..4
```

(No bound checking, so be careful!)

6. Erase at/between iterators

Erase from start to before-end:

Code:

```
1 vector<int> counts{1,2,3,4,5,6};
2 vector<int>::iterator second =
      counts.begin()+1;
3 auto fourth = second+2;
4 counts.erase(second,fourth);
5 cout << counts[0]
6      << "," << counts[1] << '\n';
```

Output:

```
1,4
```

(Also single element without end iterator.)

7. Insert at iterator

Insert at iterator: value, single iterator, or range:

Code:

```
1 vector<int> counts{1,2,3,4,5,6},
2   zeros{0,0};
3 auto after_one = zeros.begin()+1;
4 zeros.insert
5   ( after_one,
6     counts.begin()+1,
7     counts.begin()+3 );
8 cout << zeros[0] << ","
9      << zeros[1] << ","
10     << zeros[2] << ","
11     << zeros[3]
12     << '\n';
```

Output:

```
0,2,3,0
```

8. Reconstruct index

Find 'index' by getting the distance between two iterators:

Code:

```
1 vector<int> numbers{1,3,5,7,9};
2 auto it=numbers.begin();
3 while ( it!=numbers.end() ) {
4   auto d =
       distance(numbers.begin(),it);
5   cout << "At distance " << d
6        << ": " << *it << '\n';
7   it++;
8 }
```

Output:

```
At distance 0: 1
At distance 1: 3
At distance 2: 5
At distance 3: 7
At distance 4: 9
```

Algorithms

9. Reduction operation

Default is sum reduction:

Code:

```
1 #include <numeric>
2 using std::accumulate;
3   /* ... */
4     vector<int> v{1,3,5,7};
5     auto first = v.begin();
6     auto last  = v.end();
7     auto sum =
       accumulate(first,last,0);
8     cout << "sum: " << sum << '\n';
```

Output:

```
sum: 16
```

10. Reduction with supplied operator

Supply multiply operator:

Code:

```
1  using std::multiplies;
2    /* ... */
3       vector<int> v{1,3,5,7};
4       auto first = v.begin();
5       auto last  = v.end();
6       first++; last--;
7       auto product =
8         accumulate(first,last,2,
9                    multiplies<>());
10      cout << "product: " << product
         << '\n';
```

Output:

```
product: 30
```

11. Use lambda to find any of

Here is an example using any_of to find whether there is any even element in a vector:

Code:

```
1 vector<int> integers{1,2,3,5,7,10};
2 auto any_even = any_of
3   ( integers.begin(),integers.end(),
4     [=] (int i) -> bool {
5       return i%2==0; }
6     );
7 if (any_even)
8   cout << "there was an even" <<
       '\n';
9 else
10  cout << "none were even" << '\n';
```

Output:

```
there was an even
```

12. For each, very simple example

Apply something to each array element:

Code:

```
1 #include <algorithm>
2   /* ... */
3     vector<int>
       ints{2,3,4,5,7,8,13,14,15};
4     for_each(
       ints.begin(),ints.end(),
5                [] ( int i ) -> void {
6                  cout << i << '\n';
7                }
8                );
```

Output:

```
2
3
4
5
7
8
13
14
15
```

13. For any

Reduction with boolean result:

See if any element satisfies a test

Code:

```
1     vector<int>
       ints{2,3,4,5,7,8,13,14,15};
2     bool there_was_an_8 =
3       any_of(
       ints.begin(),ints.end(),
4                [] ( int i ) -> bool {
5                  return i==8;
6                }
7                );
8     cout << "There was an 8: " <<
       boolalpha << there_was_an_8 <<
       '\n';
```

Output:

```
2
3
4
5
7
8
13
14
15
```

(Why wouldn't you use a `accumulate` reduction?)

Exercise 1

Use `for_each` to sum the elements of a vector.

Hint: the problem is how to treat the sum variable. Do not use a global variable!

14. Capture by reference

Capture variables are normally by value, use ampersand for reference. This is often used in `algorithm` header.

Code:

```
1   vector<int>
       moreints{8,9,10,11,12};
2   int count{0};
3   for_each
4     (
       moreints.begin(),moreints.end(),
5       [&count] (int x) {
6         if (x%2==0)
7           count++;
8       } );
9   cout << "number of even: " <<
       count << '\n';
```

Output:

```
number of even: 3
```

15. For each, with capture

Capture by reference, to update with the array elements.

Code:

```
1     vector<int>
       ints{2,3,4,5,7,8,13,14,15};
2     int sum=0;
3     for_each(
       ints.begin(),ints.end(),
4                [&sum] ( int i ) ->
       void {
5                  sum += i;
6                }
7                );
8     cout << "Sum = " << sum << '\n';
```

Output:

```
2
3
4
5
7
8
13
14
15
```

16. Sorting

Iterator syntax:
(see later for ranges)

```
sort( myvec.begin(),myvec.end() );
```

The comparison used by default is ascending. You can specify other compare functions:

```
sort( myvec.begin(),myvec.end(),
      [] (int i,int j) { return i>j; }
    );
```