Iterators

Victor Eijkhout, Susan Lindsey

Fall 2023

last formatted: February 6, 2024

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  // stl/iter.cpp
2      vector<int> v{1,3,5,7};
3      auto pointer = v.begin();
4      cout << "we start at "
5           << *pointer << '\n';
6      ++pointer;
7      cout << "after increment: "
8           << *pointer << '\n';
9
10     pointer = v.end();
11     cout << "end is not a valid
        element: "
12          << *pointer << '\n';
13     pointer--;
14     cout << "last element: "
15          << *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 // iter/iter.cpp
2 vector<int> counts{1,2,3,4};
3 vector<int> copied(5);
4 copy( counts.begin(),counts.end(),
5       copied.begin()+1 );
6 cout << copied[0]
7      << ", " << copied[1]
8      << ".." << 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 // iter/iter.cpp
2 vector<int> counts{1,2,3,4,5,6};
3 vector<int>::iterator second =
      counts.begin()+1;
4 auto fourth = second+2;
5 counts.erase(second,fourth);
6 cout << counts[0]
7      << "," << counts[1] << '\n';
```

Output:

```
1,4
```

(Also erasing a single element without end iterator.)

7. Insert at iterator

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

Code:

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

Output:

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

8. Reconstruct index

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

Code:

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

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

Output:

```
sum: 16
```

10. Reduction with supplied operator

Supply multiply operator:

Code:

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

Output:

```
product: 30
```

11. Any of

Here is an example using `any_of` to find whether a certain element appears in a vector:

Code:

```
1 // iter/eachr.cpp
2 vector<int>
       ints{1,2,3,4,5,7,8,13,14};
3 bool there_was_an_8 =
4   std::ranges::any_of
5   ( ints,
6     [] ( int i ) -> bool {
7       return i==8;
8     }
9     );
10 cout << "There was an 8: " <<
       boolalpha << there_was_an_8 <<
       '\n';
```

Output:

```
There was an 8: true
```

12. For each, very simple example

Apply something to each array element:

Code:

```
1  // iter/eachr.cpp
2  #include <ranges>
3  #include <algorithm>
4      /* ... */
5      vector<int>
        ints{1,2,3,4,5,7,8,13,14};
6      std::ranges::for_each
7        ( ints,
8          [] ( int i ) -> void {
9            cout << i << '\n';
10         }
11         );
```

Output:

```
1
2
3
4
5
7
8
13
14
```

13. For any

Reduction with boolean result:

See if any element satisfies a test

Code:

```
1  // iter/eachr.cpp
2  #include <ranges>
3  #include <algorithm>
4      /* ... */
5      vector<int>
        ints{1,2,3,4,5,7,8,13,14};
6      std::ranges::for_each
7        ( ints,
8          [] ( int i ) -> void {
9            cout << i << '\n';
10         }
11         );
```

Output:

```
1
2
3
4
5
7
8
13
14
```

(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 // stl/printeach.cpp
2   vector<int>
      moreints{8,9,10,11,12};
3   int count{0};
4   for_each
5     (
      moreints.begin(),moreints.end(),
6       [&count] (int x) {
7         if (x%2==0)
8           ++count;
9       } );
10  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 // iter/each.cpp
2     vector<int>
       ints{2,3,4,5,7,8,13,14,15};
3     int sum=0;
4     for_each(
       ints.begin(),ints.end(),
5                [&sum] ( int i ) ->
       void {
6                  sum += i;
7                }
8                );
9     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; }
    );
```