Handout – Iterators and Algorithms

Iterator

An iterator "it" is a (generalized) pointer to an element of a container. It supports:

- Traverse the container with ++it (*increment operator*), --it, it+=k, it-=k.
- Dereference the iterator using *it (*dereference operator*), to access the element below it.

```
for (vector<int>::iterator it = a.begin();
     it != a.end(); ++it) {
    cout << *it << "\n";
}
```

behaves equivalent to:

```
for (int i = 0; i < a.size(); ++i) {
    cout << a[i] << "\n";
}
```

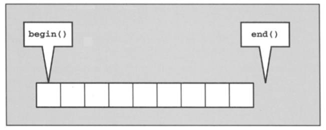
Careful:

- Don't confuse it (position) with *it (value)
- Due to operator precedence, *it.x means *(it.x), but you usually want (*it).x. Tip: it->x is equivalent to (*it).x.

Ranges

A pair of two iterators represents a range.
You can access the first element of container c with:

- c.begin() is an iterator to element c[0]
- c.end() is a "past-the-end" iterator



A range is a half-open interval: The left point is inclusive, the right point exclusive.
The length of the range is the difference between the right and the left iterator.
Examples:

- the range [a.begin(), a.end()) points to indices $\{0, 1, \ldots, n-1\}$.
- the range [a.begin()+2, a.begin()+5) points to indices $\{2, 3, 4\}$.
- the range [a.begin()+2, a.end()-3) points to indices $\{2, 3, \ldots, n-5, n-4\}$.

Algorithms

Sort

```
vector<int> a{1, 4, 5, 5, 2, 5};
sort(a.begin(), a.end());
// a: 1, 2, 4, 5, 5, 5
```

Minimum element

```
vector<int> a{1, 4, 5, 5, 2, 5};
vector<int>::iterator it =
    min_element(a.begin(), a.end());
if (it == a.end()) {
    cout << "the list is empty\n";
} else {
    cout << "the minimum value is " << *it
         << " at index " << it - a.begin()
         << '\n';
}
```

Counting an element

```
vector<int> a{1, 4, 5, 5, 2, 5};
cout << "the value 5 occurs "
     << count(a.begin(), a.end(), 5)
     << " times in a\n";
// prints "the value 5 occurs 3 times a\n"
```

Finding an element

```
vector<int> a{1, 4, 5, 5, 2, 5};
vector<int>::iterator it = find(a.begin(), a.end(), 5);
if (it == a.end()) {
    cout << "there is no 5.\n";
} else {
    cout << "the element " << *it
         << " is at index " << it - a.begin() << '\n';
}
```

Filling a vector

With the same value:

```
vector<int> a{1, 4, 5, 5, 2, 5};
fill(a.begin(), a.end, 0); // a: 0, 0, 0, 0, 0, 0
```

With values from 0 to n:

```
vector<int> a{1, 4, 5, 5, 2, 5};
iota(a.begin(), a.end(), 0); // a: 0, 1, 2, 3, 4, 5
```

Erasing elements

A specific element:

```
vector<int> a{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
a.erase(a.end() - 2);
```

A range:

```
// a: 0, 1, 2, 3, 4, 5, 6, 7, 9
a.erase(a.begin() + 3, a.begin() + 5);
// a: 0, 1, 2, 5, 6, 7, 9
```

Erase duplicates

```
vector<int> a{1, 4, 5, 5, 2, 5};
sort(a.begin(), a.end()); // needs to be sorted
// a: 1, 2, 4, 5, 5, 5
a.erase(unique(a.begin(), a.end()), a.end());
// a: 1, 2, 4, 5
// google Erase-remove idiom''
```

Passing functions as arguments

Sorting in reverse

By default sorting is done using the "<" operator. You can pass a custom "<" to sort by something else.

```
bool is_greater_than(int lhs, int rhs) {
    return lhs > rhs;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
sort(a.begin(), a.end(), is_greater_than);
// a: 8 8 7 6 5 4 3 2 2
```

Find odd elements

```
bool is_odd(int n) {
    return n % 2 == 1;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    find_if(a.begin(), a.end(), is_odd);
if (it != a.end()) {
    cout << "found an even value:" << *it << "\n";
} else {
    cout << "all values are odd\n";
}
```

Remove elements that satisfy a condition

```
bool is_odd(int n) {
    return n % 2 == 1;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    remove_if(a.begin(), a.end(), ungerade);
// a: 2 6 8 4 2 8 2 7 8
a.erase(it, a.end());
// a: 2 6 8 4 2 8
```

Lambdas

You can declare inline functions with lambdas:

```
vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    remove_if(a.begin(), a.end(),
              [](int n) { return n%2 == 1; });
// a: 2 6 8 4 2 8 2 7 8
a.erase(it, a.end());
// a: 2 6 8 4 2 8
```

More

Read https://soi.ch/wiki/stdlib/ for more detail or look at https://en.cppreference.com/w/ for more useful algorithms.