STL: Iterators

Miro Jurišić

meeroh@meeroh.org

What are iterators?

- Iterators encapsulate sequence traversal
- Separating iterators from containers
- In C, pointers are used as iterators

Basics of iterators

- Creating an iterator: from a container, from another iterator
- Sequential vs. random access
- Unidirectional vs. bidirectional
- Forward vs. reverse
- Constant vs. mutable

Creating iterators from containers

- `begin()` returns a forward iterator, pointing to the beginning of a container
- `end()` returns a forward iterator, pointing past the end of a container
- `rbegin()` returns a reverse iterator, pointing to the end of a container
- `rend()` returns a reverse iterator, pointing before the beginning of a container
- vector, deque: `begin()`, `end()`, `rbegin()`, `rend()`; random access
- string, wstring: `begin()`, `end()`, `rbegin()`, `rend()`; random access
- list: `begin()`, `end()`, `rbegin()`, `rend()`; bidirectional
- set, multiset: `begin()`, `end()`, `rbegin()`, `rend()`, `lower_bound()`, `upper_bound()`, `equal_range()`; bidirectional
- map, multimap: `begin()`, `end()`, `rbegin()`, `rend()`, `lower_bound()`, `upper_bound()`, `equal_range()`; bidirectional
- slist: `begin()`, `end()`; unidirectional

```
vector <float>        v1 (10, .1);

vector <float>::iterator i1 = v1.begin ();
vector <float>::iterator i2 = v1.end ();
vector <float>::const_iterator i3 = v1.begin ();
vector <float>::const_iterator i4 = v1.end ();
vector <float>::reverse_iterator i5 = v1.rbegin ();
vector <float>::reverse_iterator i6 = v1.rend ();
vector <float>::const_reverse_iterator i7 = v1.rbegin ();
vector <float>::const_reverse_iterator i9 = v1.rend ();

slist <float>        l1 (10, .1);

slist <float>::iterator i10 = l1.begin ();
slist <float>::iterator i11 = l1.end ();
slist <float>::const_iterator i12 = l1.begin ();
slist <float>::const_iterator i13 = l1.end ();
```

Creating iterators from other iterators

- Advance: `advance()`, `operator++()`, `operator+()`, `operator+=()`
- Retract: `advance()`, `operator--()`, `operator-()`, `operator-=()`
- Convert reverse iterators to forward iterators: `base()`
- Beware of semantics of `base()`
- Prefer pre-increment to post-increment
- Prefer equality comparison to greater-than/less-than comparison

```
vector <float>        v1 (10, .1);

vector <float>::iterator i1 = v1.begin ();
vector <float>::iterator i2 = i1 + 1;

vector <float>::const_iterator i3 = v1.begin ();
vector <float>::const_iterator i4 = i3++;

vector <float>::reverse_iterator i5 = v1.rbegin ();
vector <float>::reverse_iterator i6 = ++i5;
vector <float>::const_reverse_iterator i7 = v1.rend ();
vector <float>::const_reverse_iterator i8 = i7 - 3;

vector <float>::iterator i9 = i6.base ();

slist <float>        l1 (10, .1);

slist <float>::iterator i10 = l1.begin ();
++i10;
slist <float>::const_iterator i12 = l1.begin ();
advance (i12, 5);
```

Using iterators

- `*it`
- `it ->`
- `it []`

```
vector <float>        v1 (10, .1);

vector <float>::iterator i1 = v1.begin ();
vector <float>::iterator i2 = i1 + 1;

*i1 = .2;
if (*i2 >= 0) {
    *i2 += 1;
}

i1 [2] = .3;
```

```cpp
struct test {
    int     x;
    int     y;

    test (int, int);
};

vector <test>        v2;
v2.push_back (test (1 ,2));

vector <test>::iterator    i3 = v2.begin ();
int x = (*i3).x;
int y = i3 -> y;
```

Iterator ranges

- Iterator range: a pair of (compatible) iterators
- Describes a range of items in a container

Container manipulation using iterators

- erasing: `erase(it)`, `erase (it1, it2)`
- inserting: `insert(it, el)`
- copying: `ctor(it1, it2)`, `assign(i1, i2)`
- finding: `find()`

```
vector <float>       v1 (10, .1);

vector <float>::iterator i1 = v1.begin () + 2;
vector <float>::iterator i2 = v1.end () - 5;

v1.erase (i1);

vector <float>::iterator i3 = v1.begin () + 2;
v1.erase (i3, i2);

v1.insert (v1.end (), .2);
```

```
set <float>          s2 (v1.begin() + 1, v1.end () - 1);
set <float>::iterator    i4 = s2.find (.2);

list <float>          l3;
l3.assign (++s2.begin (), --s2.end ());
```

Iterator adapters

- An adapter is an iterator
- An adapter uses an iterator and modifies its behavior
- Example: reverse iterators
- `back_insert_iterator`, `front_insert_iterator`

```
list <float>               l1 (3, .1);
list <float>::iterator     i1 (l1.end ());
*i1 = .2;                   // runtime failure: i1 is not a valid insertion point


back_insert_iterator <list <float> >    i2  (l1);
*i1 = .2;                   // OK
*i1 = .2;                   // OK
```

Stream iterators

- Use standard streams as a sequence of items

```
vector <int>        v (
    istream_iterator <int>(cin),
    istream_iterator <int>());
```