TDDD38/726G82 - Advanced programming in C++

STL II

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Iterators

General iterations

```
for (auto&& element : c)
{
  // ...
}
```

Iterators

General iterations

```
for (auto it{c.begin()}; it != c.end(); ++it)
{
  auto&& element{*it};
  // ...
}
```

Iterators

Iterators

```
auto it{c.begin()};
auto end{c.end()};
while (it != end)
{
  // ...
  ++it;
}
```

```
auto it{c.data()};
auto end{it + c.size()};
while (it != end)
{
  // ...
  ++it;
}
```

Iterators

Iterator categories

- *ForwardIterator*
- *BidirectionalIterator*
- *RandomAccessIterator*

Iterators

InputIterator

```
std::vector<int> v{};
auto it{v.begin()};
for (int i{0}; i < 10; ++i)
{
  *it++ = i;
}
```

Iterators

OutputIterator

- *InputIterator*
  - \+ Can access elements in container
  - − Cannot add elements to container

Iterators

OutputIterator

- *InputIterator*
  - + Can access elements in container
  - − Cannot add elements to container
- *OutputIterator*
  - + Can add elements to container
  - − Cannot access elements in container

Iterators

`std::insert_iterator`

```
std::vector<int> v{};
auto it{std::inserter(v, v.end())};
for (int i{0}; i < 10; ++i)
{
  *it++ = i;
}
```

Iterators

`std::insert_iterator`

```
std::vector<int> v{};
auto it{std::inserter(v, v.end())};
for (int i{0}; i < 10; ++i)
{
  *it = i;
}
```

Iterators

`std::insert_iterator`

```
std::vector<int> v{};
auto it{std::inserter(v, v.end())};
for (int i{0}; i < 10; ++i)
{
  it = i;
}
```

Iterators

`std::insert_iterator`

```
std::vector<int> v{};
auto it{std::inserter(v, v.end())};
for (int i{0}; i < 10; ++i)
{
  v.insert(v.end(), i);
}
```

Iterators

Other output iterators

- `std::back_inserter`
- `std::front_inserter`

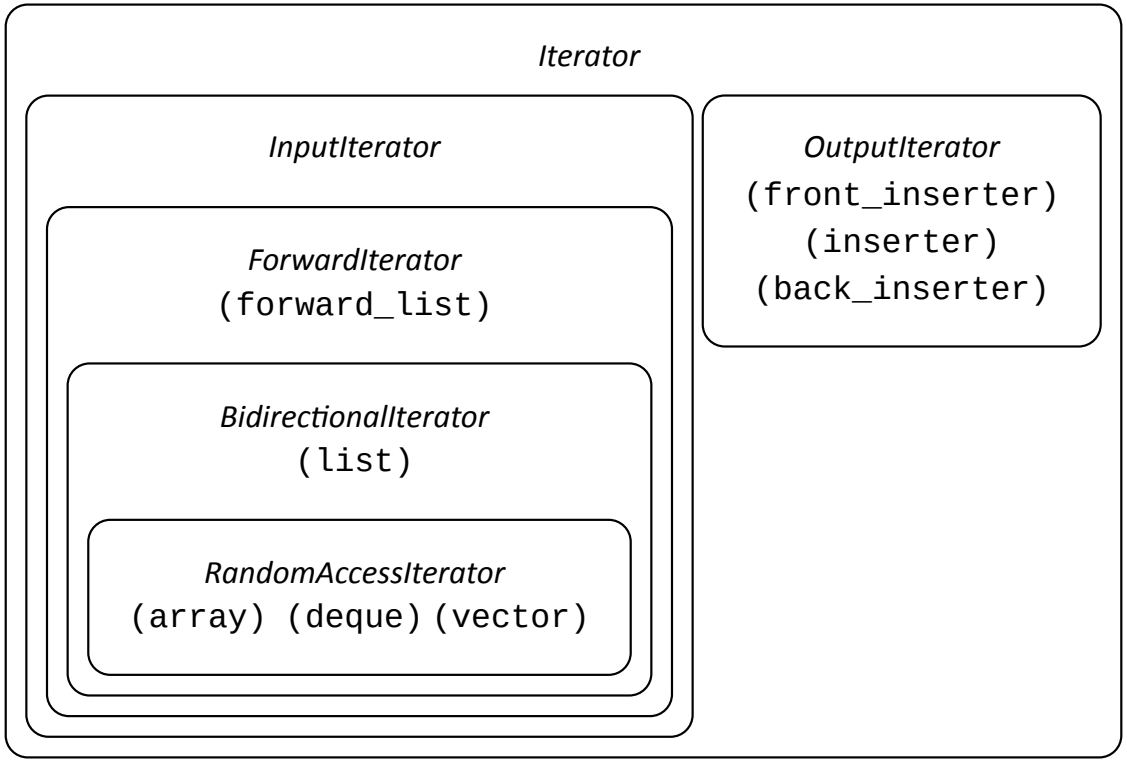
Iterators

Other output iterators

- `std::back_inserter`
- `std::front_inserter`
- These only need to know the container, since their insertion positions are fixed

Iterators

Iterator hierarchy



Iterators

What will be printed?

```
int main()
{
    std::vector<int> v {1, 3};
    *std::back_inserter(v)++ = 7;

    int value { ++(*v.begin()) };
    std::inserter(v, v.begin() + 1) = value;
    for (int i : v)
    {
        std::cout << i << " ";
    }
}
```

LINKÖPING UNIVERSITY

Associative Containers

`std::set`

```
std::set<int> set{};
```

Associative Containers

`std::set`

{}

```
std::set<int> set{};
```

Associative Containers

`std::set`

{}

```
set.insert(4);
```

Associative Containers

`std::set`

{4}

```
set.insert(4);
```

Associative Containers

`std::set`

{4}

```
set.insert(3);
```

Associative Containers

`std::set`

{3, 4}

```
set.insert(3);
```

Associative Containers

`std::set`

{3, 4}

```
set.insert(5);
```

Associative Containers

`std::set`

```
{3, 4, 5}
```

```
set.insert(5);
```

Associative Containers

`std::set`

```
{3, 4, 5}
```

```
set.insert(1);
```

Associative Containers

`std::set`

{1, 3, 4, 5}

```
set.insert(1);
```

Associative Containers

`std::set`

{1, 3, 4, 5}

```
set.insert(2);
```

Associative Containers

`std::set`

```
{1, 2, 3, 4, 5}
```

```
set.insert(2);
```

Associative Containers

`std::set`

```
{1, 2, 3, 4, 5}
```

```
set.erase(3);
```

Associative Containers

`std::set`

```
{1, 2, 4, 5}
```

```
set.erase(3);
```

Associative Containers

`std::set`

- insertion: $O(\log n)$
- deletion: $O(\log n)$
- lookup: $O(\log n)$

Associative Containers

Example

```
#include <set>
// ...
int main()
{
  std::set<std::string> words{};
  std::string str;
  while (cin >> str)
  {
    set.insert(str);
  }
  for (auto const& word : words)
  {
    cout << word << endl;
  }
}
```

Associative Containers

`std::map`


```
std::map<std::string, int> map{};
```

Associative Containers

`std::map`


```
map["c"] = 3;
```

Associative Containers

`std::map`

"c"	3

```
map["c"] = 3;
```

Associative Containers

`std::map`

"c"	3

```
map["a"] = 1;
```

Associative Containers

`std::map`

"a"	1
"c"	3

```
map["a"] = 1;
```

Associative Containers

`std::map`

"a"	1
"c"	3

```
map["d"] = 4;
```

Associative Containers

`std::map`

"a"	1
"c"	3
"d"	4

```
map["d"] = 4;
```

Associative Containers

`std::map`

"a"	1
"c"	3
"d"	4

```
map["b"] = 2;
```

Associative Containers

`std::map`

"a"	1
"b"	2
"c"	3
"d"	4

```
map["b"] = 2;
```

Associative Containers

`std::map`

- insertion: $O(\log n)$
- deletion: $O(\log n)$
- lookup: $O(\log n)$

Associative Containers

Example

```
#include <map>
// ...
int main()
{
  std::map<std::string, int> words{};
  std::string str;
  while (cin >> str)
  {
    words[str]++;
  }
  for (std::pair<std::string, int> const& p : words)
  {
    cout << p.first << ": " << p.second << endl;
  }
}
```

Associative Containers

Variants

- `multi*`
- `unordered_*`

Associative Containers

Variants

- `multi*`
  - `std::multiset`
  - `std::multimap`
- `unordered_*`

Associative Containers

Variants

- `multi*`
- `unordered_*`
  - `std::unordered_set`
  - `std::unordered_map`
  - `std::unordered_multiset`
  - `std::unordered_multimap`

LINKÖPING UNIVERSITY

Container Adaptors

`std::stack`

```
template <typename T,
          typename Container = std::deque<T>>
class stack;
```

```
std::stack<int> st{};
st.top();  // top of stack
st.push(); // push to stack
st.pop();  // pop the stack
```

Container Adaptors

`std::queue`

```
template <typename T,
          typename Container = std::deque<T>>
class queue;
```

```
std::queue<int> q{};
q.front(); // front of the queue
q.back();  // back of the queue
q.push();  // add element to back of queue
q.pop();   // pop first element of the queue
```

Container Adaptors

`std::priority_queue`

```
template <typename T,
          typename Container = std::vector<T>,
          typename Compare = std::less<T>>
class priority_queue;
```

```
std::priority_queue<int> pq{};
pq.top();  // get the largest value
pq.push(); // add an element
pq.pop();  // extract the largest value
```

Container Adaptors

Example

```
int main()
{
  std::priority_queue<float, std::greater<float>> q{};

  float value;
  while (cin >> value)
  {
    q.push(value);
  }

  float sum{0.0};
  while (!q.empty())
  {
    sum += q.top();
    q.pop();
  }

  cout << sum << endl;
}
```

LINKÖPING UNIVERSITY

Lambda Functions

Possible implementation of `std::less`

```
template <typename T>
struct less
{
  bool operator()(T const& lhs,
                  T const& rhs)
  {
    return lhs < rhs;
  }
};
```

```
int main()
{
  less<int> obj{};

  // we can use the function call
  // operator  to treat this object
  // as a function
  cout << obj(1, 2) << endl;
}
```

Lambda Functions

First-class functions

```
template <typename Function>
auto perform(Function f) -> decltype(f())
{
  return f();
}
```

Lambda Functions

First-class functions

```
struct my_function
{
  int operator()()
  { return 1; }
};

int main()
{
  my_function f{};
  perform(f);
}
```

Lambda Functions

Example

```
template <typename Container,
          typename Compare>
bool is_sorted(Container const& c,
               Compare const& comp)
{
  auto it{c.begin()};
  auto prev{it++};
  for (; it != c.end(); ++it)
  {
    if (!comp(*prev, *it))
      return false;
    prev = it;
  }
  return true;
}
```

```
int main()
{
  std::vector<int> v{1,2,3,4};
  std::deque<int>  d{3,2,1,0};

  std::less<int>    lt{};
  std::greater<int> gt{};

  cout << is_sorted(v, lt);
  cout << is_sorted(d, gt);
}
```

Lambda Functions

Lambda expressions

```
std::vector<int> v{10, -1, 12};

is_sorted(v, [](int a, int b) -> bool
             {
               return abs(a - 10) < abs(b - 10);
             });
```

Lambda Functions

Lambda expressions

```
[](int a, int b) -> bool
{
  return abs(a - 10) < abs(b - 10);
}
```

```
struct my_lambda
{
  bool operator()(int a, int b)
  {
    return abs(a - 10) < abs(b - 10);
  }
};
```

Lambda Functions

Captures

```
std::vector<int> v{10, -1, 12};
int x{10};

auto comp{[x](int a, int b) -> bool
          {
            return abs(a - x) < abs(b - x);
          }};

is_sorted(v, comp);
```

Lambda Functions

Captures

```
[x](int a, int b) -> bool
{
  return abs(a - x) < abs(b - x);
}
```

```
struct my_lambda
{
  my_lambda(int x) : x{x} { }
  bool operator()(int a, int b)
  {
    return abs(a - x) < abs(b - x);
  }
private:
  int const x;
};
```

Lambda Functions

Captures

```
[&x](int a, int b) -> bool
{
  return abs(a - x) < abs(b - x);
}
```

```
struct my_lambda
{
  my_lambda(int& x) : x{x} { }
  bool operator()(int a, int b)
  {
    return abs(a - x) < abs(b - x);
  }
private:
  int& x;
};
```

Lambda Functions

Captures

```
[x = 10](int a, int b) -> bool
{
  return abs(a - x) < abs(b - x);
}
```

```
struct my_lambda
{
  my_lambda() : x{10} { }
  bool operator()(int a, int b)
  {
    return abs(a - x) < abs(b - x);
  }
private:
  int const x;
};
```

Lambda Functions

Captures

```
[x, &y, z = 10](/* function parameters */)
{
  // ...
}
```

Lambda Functions

mutable

```
int x{};
auto f = [x]() { x = 1; };
```

Lambda Functions

mutable

```
int x{};
auto f = [x]() mutable { x = 1; };
```

Lambda Functions

Special captures

```
int global{1};
int main()
{
  int x{2};
  int y{3};
  auto f{[&]()
         {
           return x + y + global;
         }};
  f(); // will return 6
  y = -3;
  f(); // will return 0
}
```

Lambda Functions

Special captures

```
int global{1};
int main()
{
  int x{2};
  int y{3};
  auto f{[=]()
         {
           return x + y + global;
         }};
  f(); // will return 6
  y = -3;
  f(); // will return 6
}
```

Lambda Functions

Mixing captures

```
[=, &x](/* parameters */)
{
  // ...
}
```

Lambda Functions

What will be printed?

```
int main()
{
  auto f = [n = 0]() mutable { return n++; };
  auto g = f;
  cout << f() << ' ';
  cout << f() << ' ';
  cout << g() << endl;
}
```

www.liu.se

LINKÖPING
UNIVERSITY