Associative Containers & Iterators

Cristian Cibils
(ccibils@stanford.edu)

Recap of Sequence Containers

- Sequence containers store data stored in order
- STL and Stanford sequence containers are similar but have a few key differences

Associative Containers

- Like Sequence Containers, Associative containers store data
- Unlike Sequence Containers, Associative containers have no idea of an ordering
- Instead, based on a **key**
- We will look at a few Associative Containers today
  - Map
  - Set

STL <map>

- Methods are the same as the Stanford Map except for some syntax differences
  - If you want to see a complete list of methods, google search std::map or check out http://www.cplusplus.com/reference/map/map
- Let's see an example using a map (Map.pro)

STL <set>

- Methods are the same as the Stanford Set except for some syntax differences
  - If you want to see a complete list of methods, google search std::set or check out http://www.cplusplus.com/reference/set/set/
- Let's see an example using a set (Set.pro
- Key point, a set is just a specific case of a map

Announcements

- Reminder: Assignment 1 due next Tuesday at midnight

Iterators

- How do we iterate over associative containers?

Iterators

- How do we iterate over associative containers?
  - Not an easy question to answer
  - We first need to look at some history of C++

Versions of C++

- 1983-1998: Prehistoric C++
  - This was the original form of C++. There was no document describing what C++ was. Any code which would compile and run was valid C++
  - This was fine, but many people wanted a standard
- 1998: C++98
  - The C++98 standard is published, which details exactly what valid C++ code is.
- 2003: C++03
  - Minor changes to the rules governing how programs run, but effectively the same as C++98

Versions of C++

- 2011: C++11!
  - C++11 was the largest change to the C++ language since its creation
  - Clarified a lot of things in the old standard
  - Added **tons** of new features
    - Multithreading and atomic types
    - Type inference
    - Lambdas
    - Range based for
    - `constexpr`
    - New initialization syntax
  - Long story short, C++11 has a lot of cool stuff in it!

Versions of C++

C++03	C++11

`map<string, int> map;`
 	
`map<string, int>::iterator i;`
`map<string, int>::iterator end`
`  = map.end();`
 	
`for( i = address_book.start();`
`     i != end; ++i) {`
`    cout  << (*i).first`
`          << " " << (*i).second`
`          << endl;`
`}`
	`map<string, int> map;`
	
	`for (auto& a : map) {`
	`    cout  << a.first << " "`
	`          << a.second << endl;`
	`}`

Versions of C++

- If we have C++11, why learn about iterators?

Versions of C++

- If we have C++11, why learn about iterators?
  - Not all compilers support C++11

Versions of C++

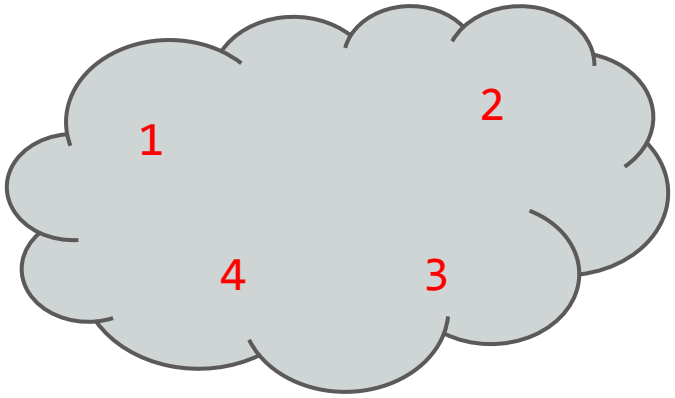
- If we have C++11, why learn about iterators?
  - Not all compilers support C++11
  - Iterators are used for more than just iterating as we will see when we start talking about ranges today and algorithms next week

Iterators

Iterators allow a programmer to iterate over all the values in any container whether it is ordered or not

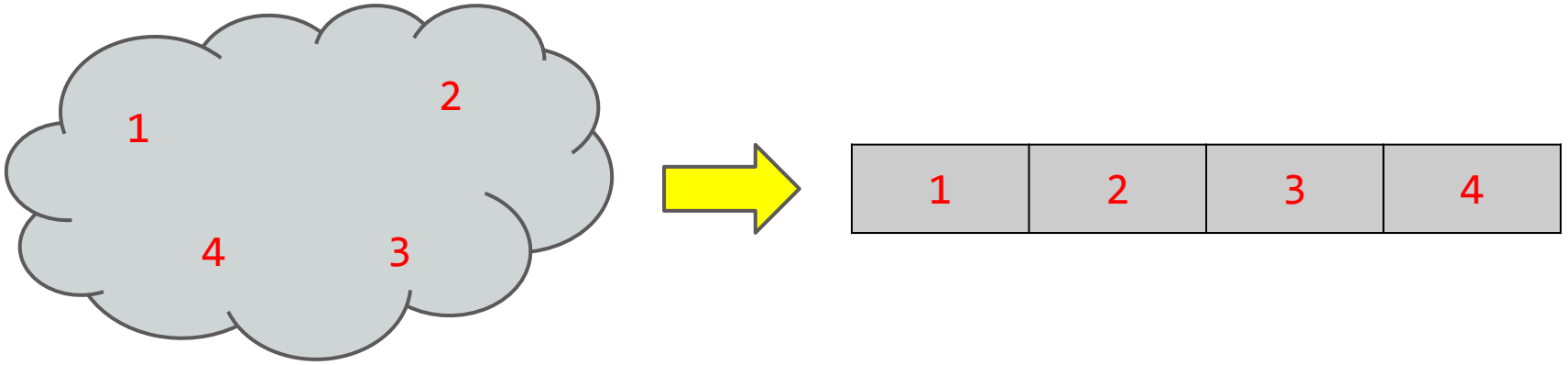
Iterators

- Let's first try and get a conceptual model of what an iterator is
- Say that we have a set of integers called mySet



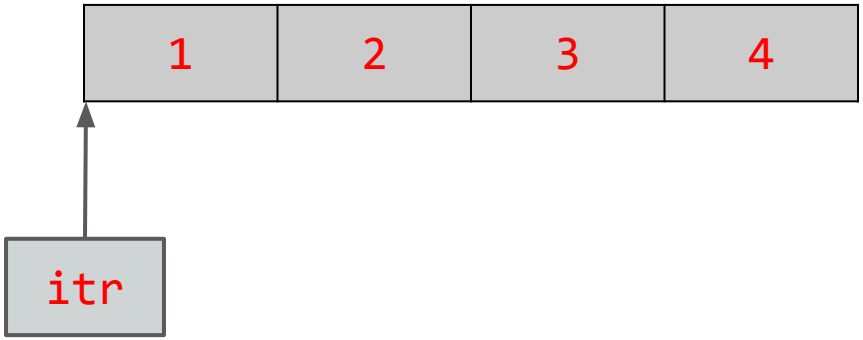
Iterators

- Let's first try and get a conceptual model of what an iterator is
- Iterators allow us to view a non-linear collection in a linear manner



Iterators

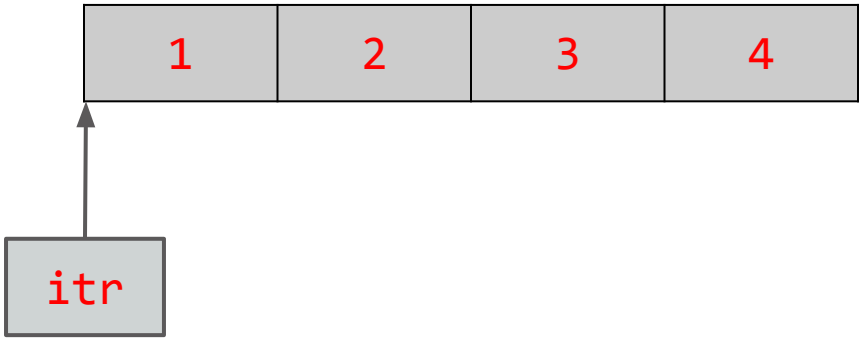
- Let's first try and get a conceptual model of what an iterator is
- We can construct an iterator 'itr' to point to the first element in the set




```
set<int>::iterator itr = mySet.begin();
```

Iterators

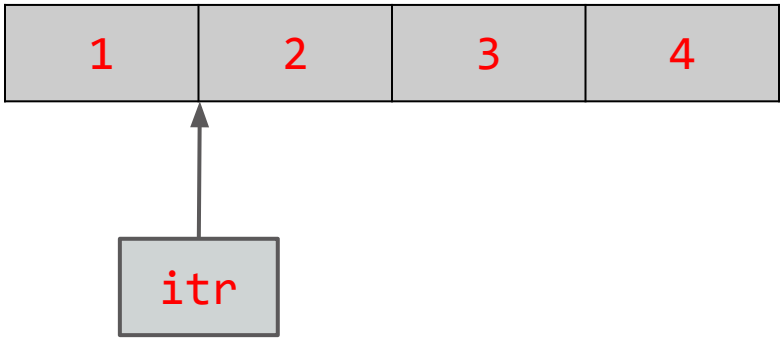
- Let's first try and get a conceptual model of what an iterator is
- We can get the value of an iterator by using the **dereference** operator *




```
cout << *itr << endl; //prints 1
```

Iterators

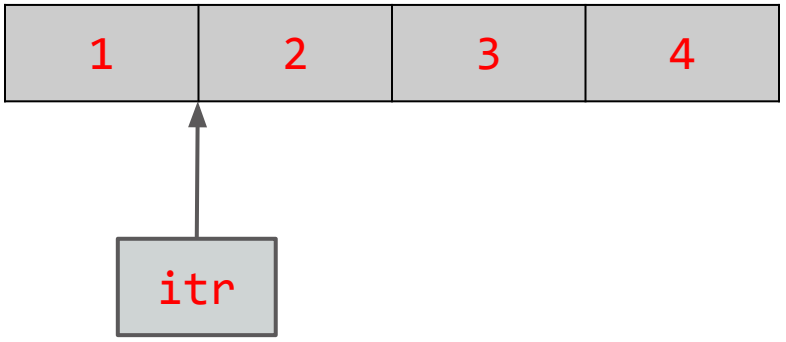
- Let's first try and get a conceptual model of what an iterator is
- We can advance our iterator




```
++itr;
```

Iterators

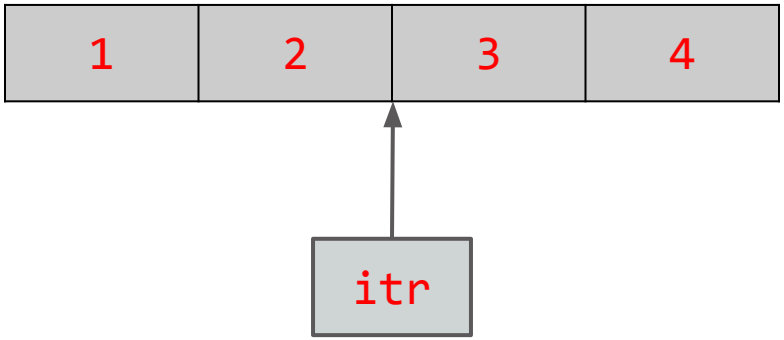
- Let's first try and get a conceptual model of what an iterator is
- We can keep advancing and dereferencing




```
cout << *itr << endl; //prints 2
```

Iterators

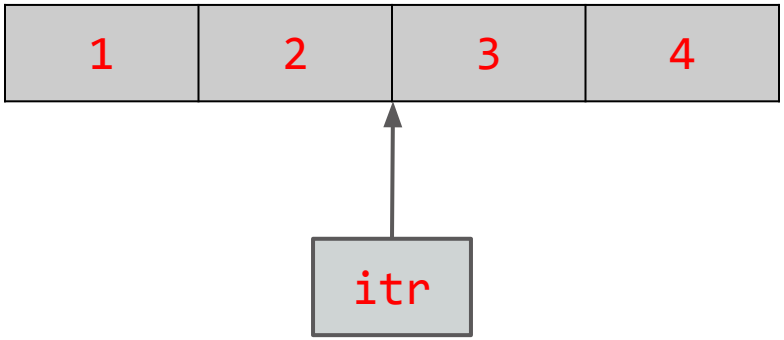
- Let's first try and get a conceptual model of what an iterator is
- We can keep advancing and dereferencing




```
++itr;
```

Iterators

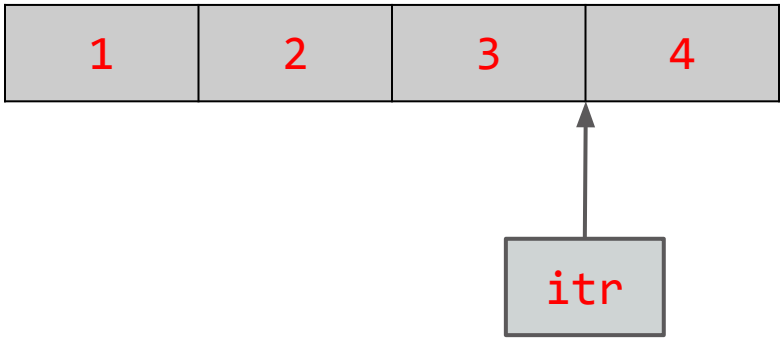
- Let's first try and get a conceptual model of what an iterator is
- We can keep advancing and dereferencing




```
cout << *itr << endl; //prints 3
```

Iterators

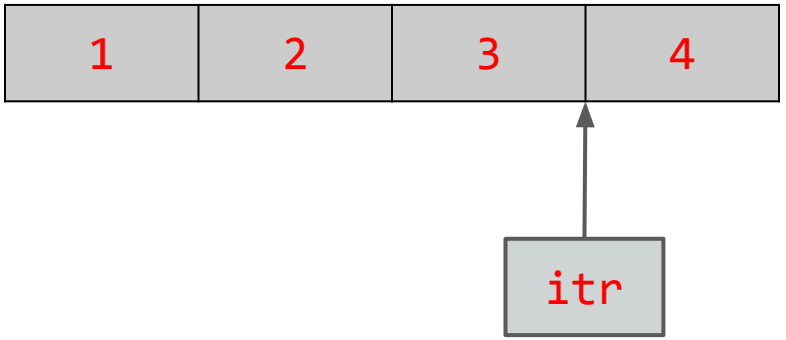
- Let's first try and get a conceptual model of what an iterator is
- We can keep advancing and dereferencing




```
++itr;
```

Iterators

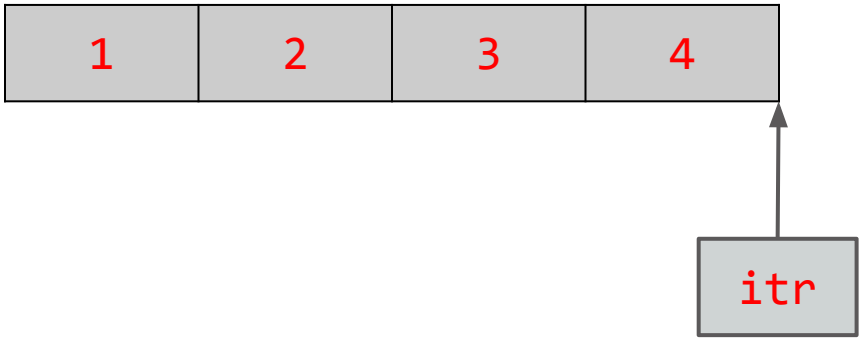
- Let's first try and get a conceptual model of what an iterator is
- We can keep advancing and dereferencing




```
cout << *itr << endl; //prints 4
```

Iterators

- Eventually we reach the end of the container
- We can check if we have reached the end by comparing our iterator to `.end()`




```
if (itr == mySet.end()) return;
```

Iterators

To Recap, four essential iterator operators:

- Create an iterator
- **Dereference** an iterator and read the value it's currently looking at
- **Advance** an iterator
- Compare an iterator against another iterator (especially one from the `.end()` method)

Iterators

Let’s Do Some Examples
(BasicIterator.pro)

Versions of C++

- If we have C++11, why learn about iterators?
  - Not all compilers support C++11
  - Iterators are used for more than just iterating as we will see when we start talking about ranges today and algorithms next week

Versions of C++

- If we have C++11, why learn about iterators?
  - Not all compilers support C++11
  - Iterators are used for more than just iterating as we will see when we start talking about ranges today and algorithms next week

Other Uses for Iterators

STL containers often use iterators to specify individual elements inside a container.

```
vector<int> v;
for (int i = 0; i < 10; i++) {
   v.push_back(i);
}
v.erase(v.begin() + 5, v.end());
// v now contains 0, 1, 2, 3, 4
```

Other Uses for Iterators

Iterators don’t always have to iterate through an entire container

```
set<int>::iterator i = mySet.begin();
set<int>::iterator end = mySet.end();
while (i != end) {
   cout << *i << endl;
   ++i;
}
```

Other Uses for Iterators

Here is code that will iterate through all elements **greater than or equal to** 7 and **less than** 26

```
set<int>::iterator i = mySet.lower_bound(7);
set<int>::iterator end = mySet.lower_bound(26);
while (i != end) {
   cout << *i << endl;
   ++i;
}
```

Other Uses for Iterators

Note that we can iterate through various ranges of numbers simply by choosing different values of begin and end

	[a, b]	[a, b)	(a, b]	(a, b)
begin	lower_bound(a)	lower_bound(a)	upper_bound(a)	upper_bound(a)
end	upper_bound(b)	lower_bound(b)	upper_bound(b)	lower_bound(b)

Other Uses for Iterators

Let’s code up some examples

(IteratorRanges.pro)

Iterating through maps

- All of our iterator examples involved set iterators, but (almost) all C++ collections have iterators
- Sequence Container iterators are straightforward
- Maps are a little more complicated

The Pair Class

- A pair is simply two objects bundled together
- Syntax is the following:

```
pair<string, int> p;
p.first = “phone number”;
p.second = 8675309;
```

Iterating through maps

- When iterating through maps, dereferencing returns a pair containing the key and the value of the current element

```
map<int, int> m;
map<int, int>::iterator i = m.begin();
map<int, int>::iterator end = m.end();
while (i != end) {
   cout << (*i).first << (*i).second << endl;
   ++i;
}
```

Multiset

- Sets store unique elements
- If you want to store multiple copies of an element, use a multiset
- Almost all methods are the same

```
multiset<int> myMSet;
myMSet.insert(3);
myMSet.insert(3);
cout << myMSet.count(3) << endl; //prints 2
```

Closing Notes

- Iterators are used everywhere in C++ code
- When you first look at a C++ style iterator, you may find yourself missing foreach, but iterators offer a lot more
- Iterator ranges are just the start. When we start talking about <algorithm> We'll see just how useful iterators can be
- Don't forget assignment one!