Iterator

Dmitri Nesteruk
QUANTITATIVE ANALYST

@dnesteruk   http://activemesa.com

Overview

Motivation

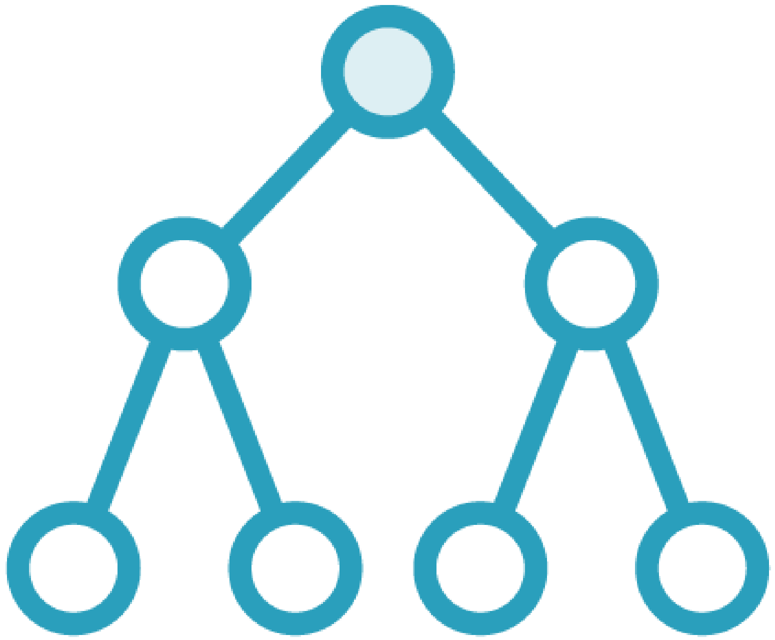
Iterators in the Standard Library

Iterator Requirements

Binary Tree Iterator

Boost Iterator Façade

Motivation



Iteration (traversal) is a core functionality of various data structures

An *iterator* is a class that facilitates the traversal

- Keeps pointer to an element
- Knows how to move to a different element

Iterator types

- Forward (e.g., on a list)
- Bidirectional (e.g., on a doubly linked list)
- Random access (e.g., on a vector)

Iterator

An object that facilitates the traversal of a data structure.

Iterator Requirements

Container member functions

`beginXxx()`
points to the first element in the container; if empty, is equal to `endXxx()`

`endXxx()`
points to the element immediately after the last element

Facilitate use of standard algorithms

Allow the use of range-based for loop
`for (auto& x : my_container)`

Different names for different iterators

Iterator operators

`operator !=`
must return `false` if two iterators point to the same element

`operator *` (dereferencing)
must return a reference to (or a copy of) the data the iterator points to

`operator ++`
gets the iterator to point to the next element

Additional operators as required (e.g., `operator--`, arithmetic, etc.)

Summary

An iterator specifies how you can traverse an object

Typically needs to support comparison (!=), advancing (++) and dereferencing (*)

- May support other things, e.g., arithmetic, operator --, etc.

Can have many different iterators (reverse, const, etc.)

- Default one returned in begin()/end()

Iterators cannot be recursive ☹