UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 11: More on Standard Template Library (STL)

mihasan@liverpool.ac.uk

Previous lecture

- The Standard Template Library (STL) was introduced and its components were discussed.
- Different types of containers were discussed.
- Vectors, deques, and lists were discussed.
- Maps were discussed.

This lecture

- What is covered in this lecture?

  Iterators and algorithms in the Standard Template Library (STL)

- Why it is covered?

  Combining Iterators and algorithms with containers realises the full potential of STL

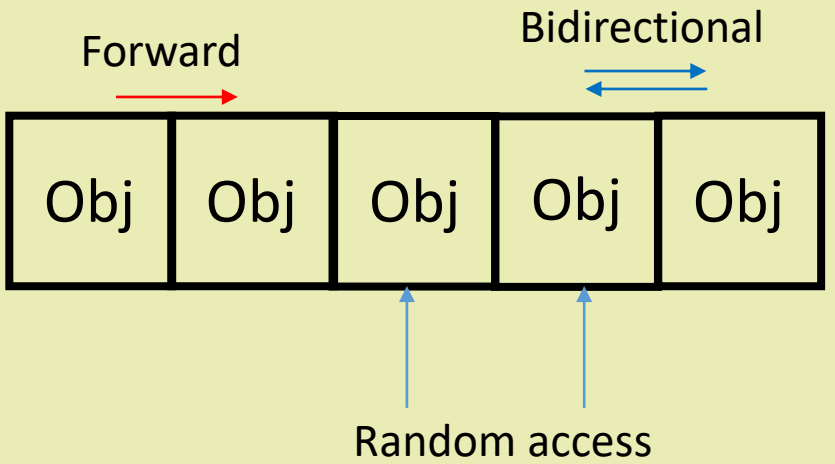
- How are topics covered in this lecture:

  6 source codes.

Iterators

- Iterators are objects used to access elements of containers.
- Can be included in any code using `#include <iterator>`
- Reference: http://www.cplusplus.com/reference/iterator/
- There are 5 categories of iterators with different capabilities:

1. **Input**: read element multiple times but can't write.
2. **Output**: only write an element and can do it once.
3. **Forward**: read and write elements while moving toward the last element.
4. **Bidirectional**: read and write elements while moving toward the last or the first element.
5. **Random access**: read and write elements anywhere in the container.



Iterators

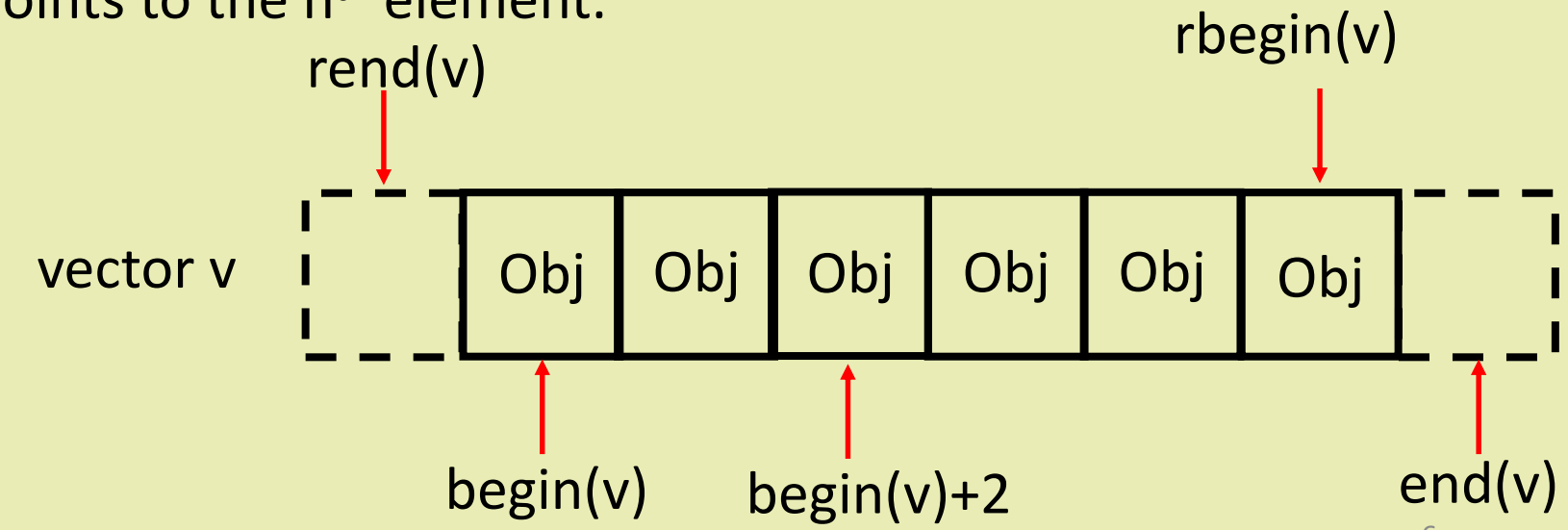
- If we have an iterator **`i`** and a variable **`x`** then:

	Output	Input	Forward	Bidirectional	Random
Read		`x = *i`	`x = *i`	`x = *i`	`x = *i`
Write	`*i = x`		`*i = x`	`*i = x`	`*i = x`
Move	++	++	++	++, --	++, --, +, - ,-=, +=
Compare		== , !=	== , !=	== , !=	== , !=, <, >, <=, >=

- The keyword "`auto`" can most of the time identify the right type of iterator.
- Iterators provide higher functionality in comparison to raw pointers.
- Go to L11D1.cpp

Vector iterators

- Vector iterators are iterator-specific to vector-type containers.
- The datatype of vector-specific iterator is "`vector<template>::iterator`"
- Iterators "begin" and "end" give access to first and one past last element.
- Iterator "`begin + n`" points to the n^{th} element.
- Go to L11D2.cpp



Stream iterators

- Stream iterators are stand-alone iterators which iterate on input/output streams.
- Header: `#include <iterator>`
- Ref: http://www.cplusplus.com/reference/iterator/istream_iterator/
- Ref: http://www.cplusplus.com/reference/iterator/ostream_iterator/
- They behave similar to input iterators and output iterators.
- For input stream (using cin), the stream iterator is **`istream_iterator`**. For output streams (using cout), the stream operator is **`ostream_iterator.`**
- Go to L11D3.cpp

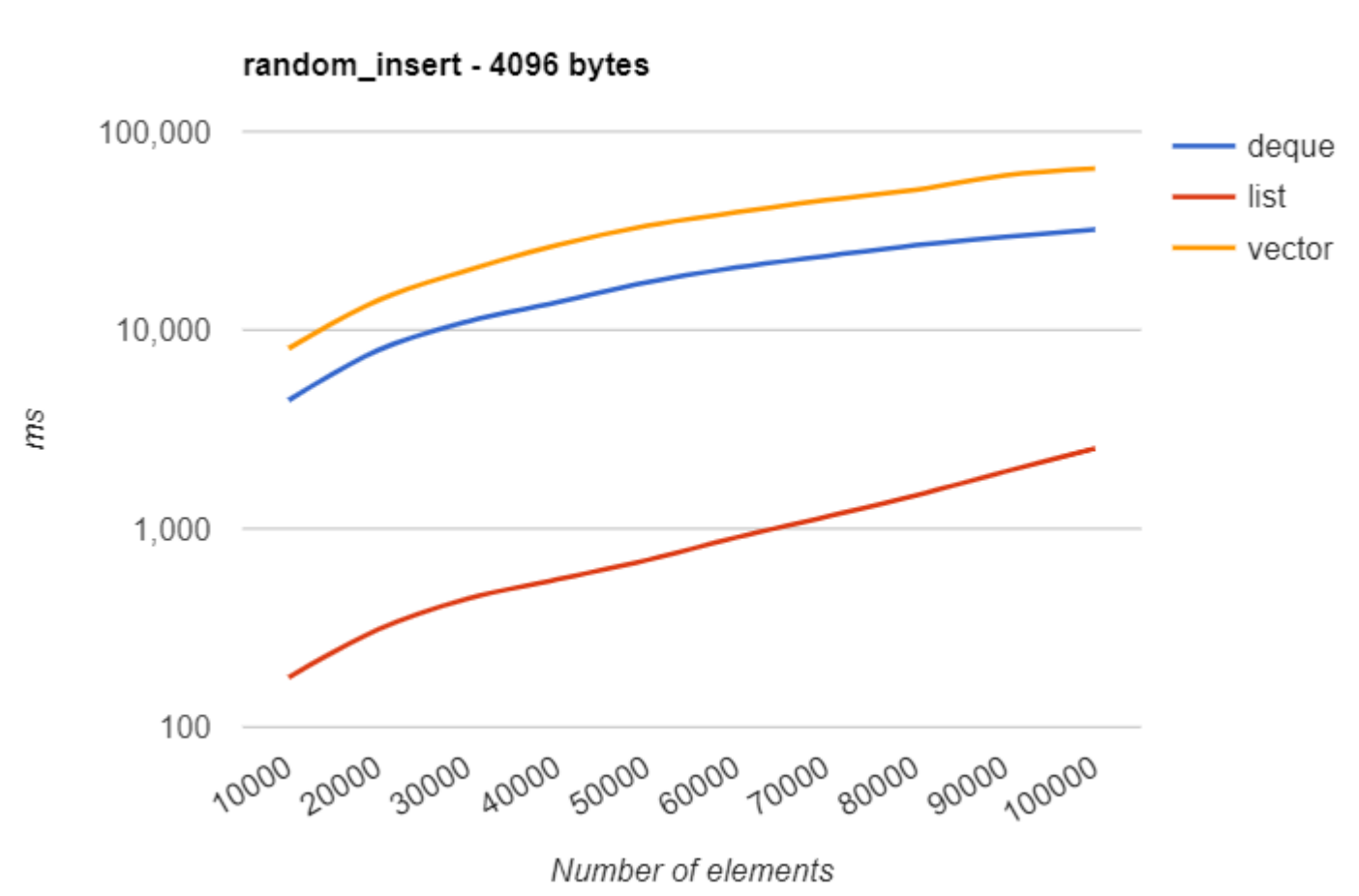
Insert iterators

- Insert iterators are stand-alone iterators that can be used to break insertion/deletion rules for certain containers (including vector, deque, and list).
- Header: `#include <iterator>`
- They are used to specify a location in the container to insert a new element.
- There are three types of insert iterators:

Iterator	Vector-equivalent function
back_insert_iterator <container <T>>	push_back()
front_insert_iterator <container <T>>	push_front()
insert_iterator <container <T>>	insert()

- QUESTION: What is the point of having insertion rules for containers then?

Insertion benchmark



https://baptiste-wicht.com/posts/2012/12/cpp-benchmark-vector-list-deque.html#

Practical note:

Use the most suitable container for the purpose of your code

- Go to L11D4.cpp

Smart pointers

- Smart pointers are stand-alone iterators that behave like a pointer but is optimised for dynamic memory allocation.
- Can be included in any code using `#include <memory>`
- Reference: http://www.cplusplus.com/reference/memory/
- Most of smart pointers were introduced in C++11.
- A noticeable feature when using smart pointers is that "**`delete`**" is no longer required.

Smart pointers types

- **unique_ptr** : A smart pointer that contains the only copy of the address of a given location in memory, such that the address (i.e. the ownership) can be moved to another pointer without copying.
- **shared_ptr** : A smart pointer similar to raw pointers, except that it counts the number of pointers holding the same address.
- **weak_ptr** : A special case of **shared_ptr** where it has the same address of that of **shared_ptr** without being included in reference counting.
- Go to L11D5.cpp

Good practice note: User smart pointers mainly when ownership matters.

Algorithms

- Algorithms are function templates that act on containers and provide means for various operations for the contents of the containers.
- Can be included in any code using `#include <algorithm>`
- Reference: http://www.cplusplus.com/reference/algorithm/
- Examples of these algorithms include sorting, searching, finding minimum/maximum, etc.

Functors

- Functors are classes where the objects behave as functions.
- They are used in many occasions in STL library.
- Can be included in any code using `#include <functional>`
- The basic idea behind functor is to overload the call operator, such that:

  MyFunctor (1) ⟵ MyFunctor.operator() (1)

- Reference: http://www.cplusplus.com/reference/functional/
- Go to L11D6.cpp

Summary

- Iterators were introduced an the different types of iterators were discussed.
- Stream iterators and insert iterators were discussed.
- Smart pointers were discussed.
- Algorithms and Functors were defined and discussed.