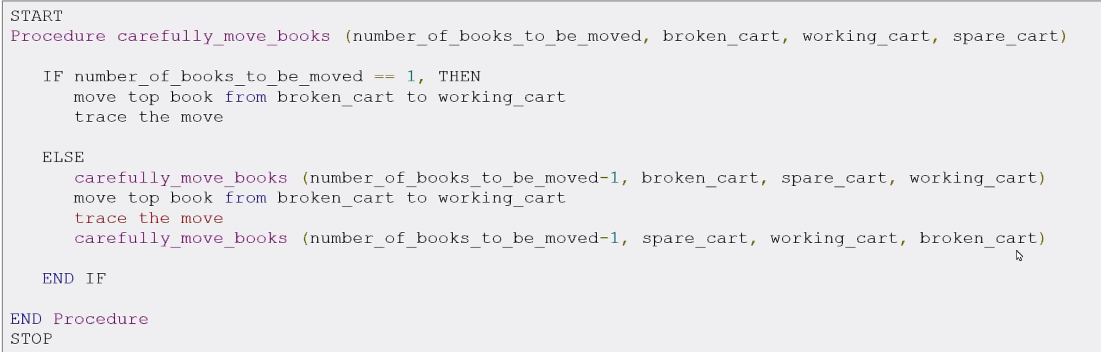
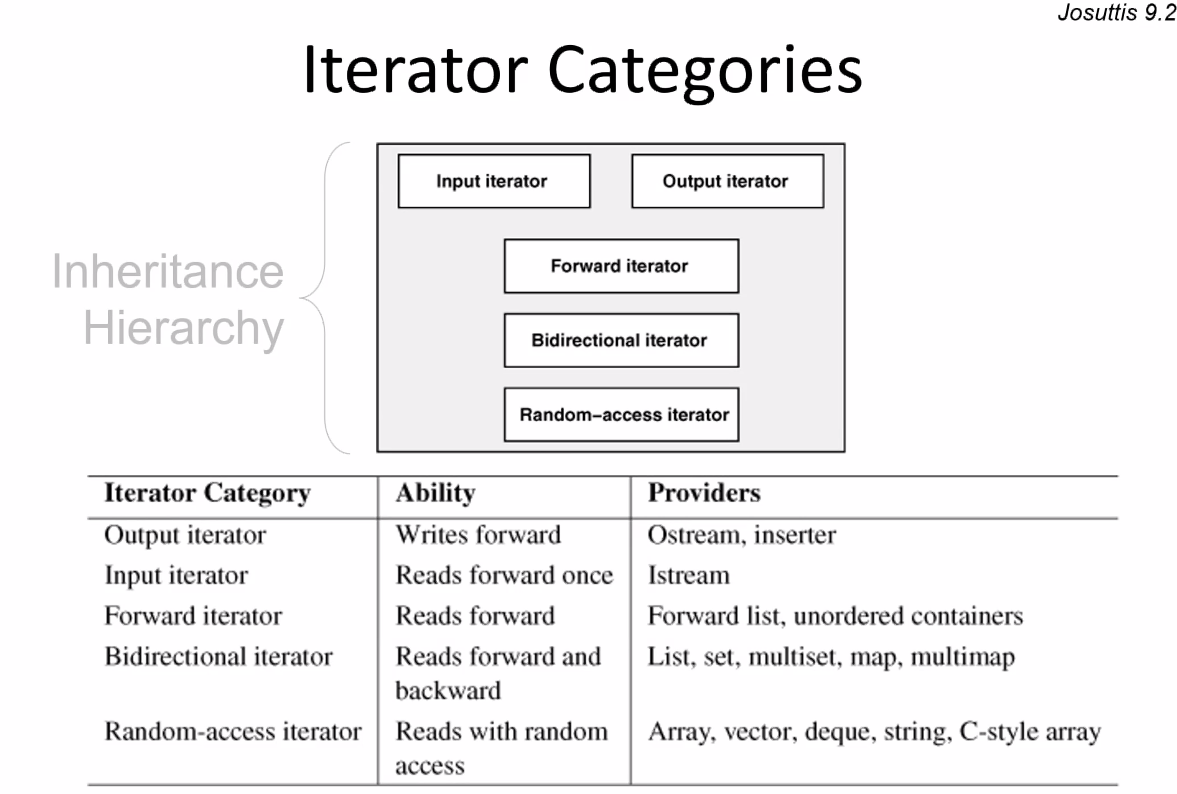
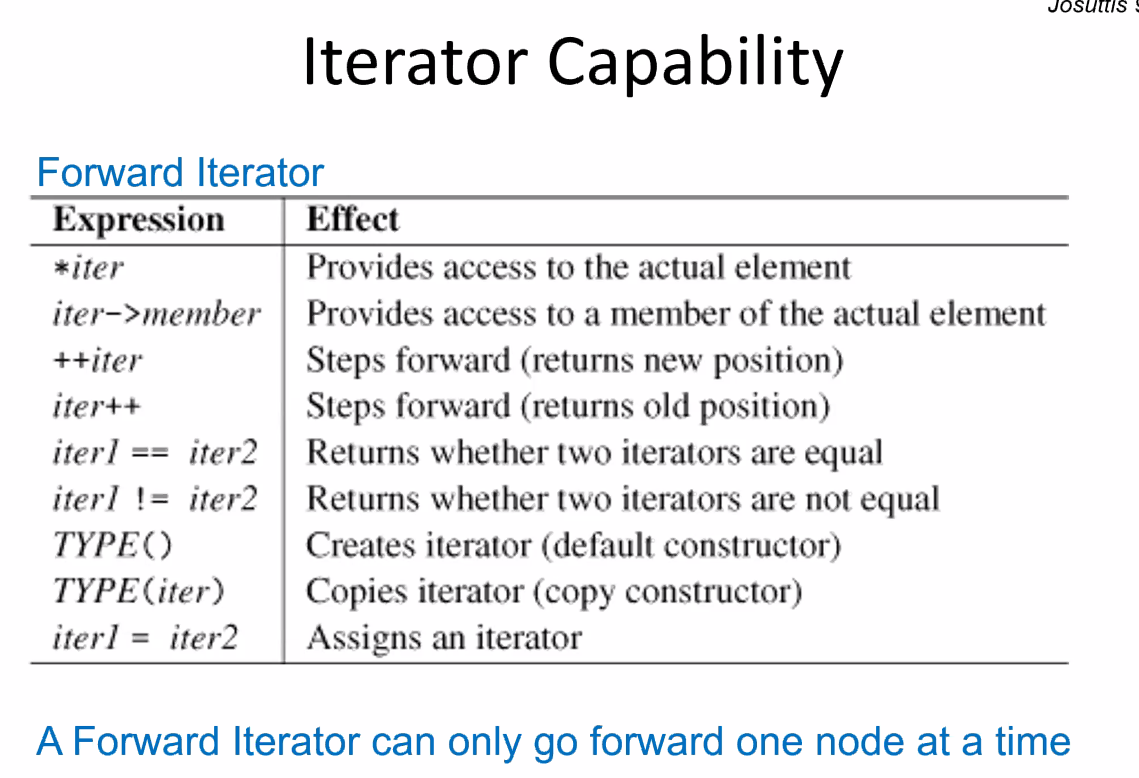
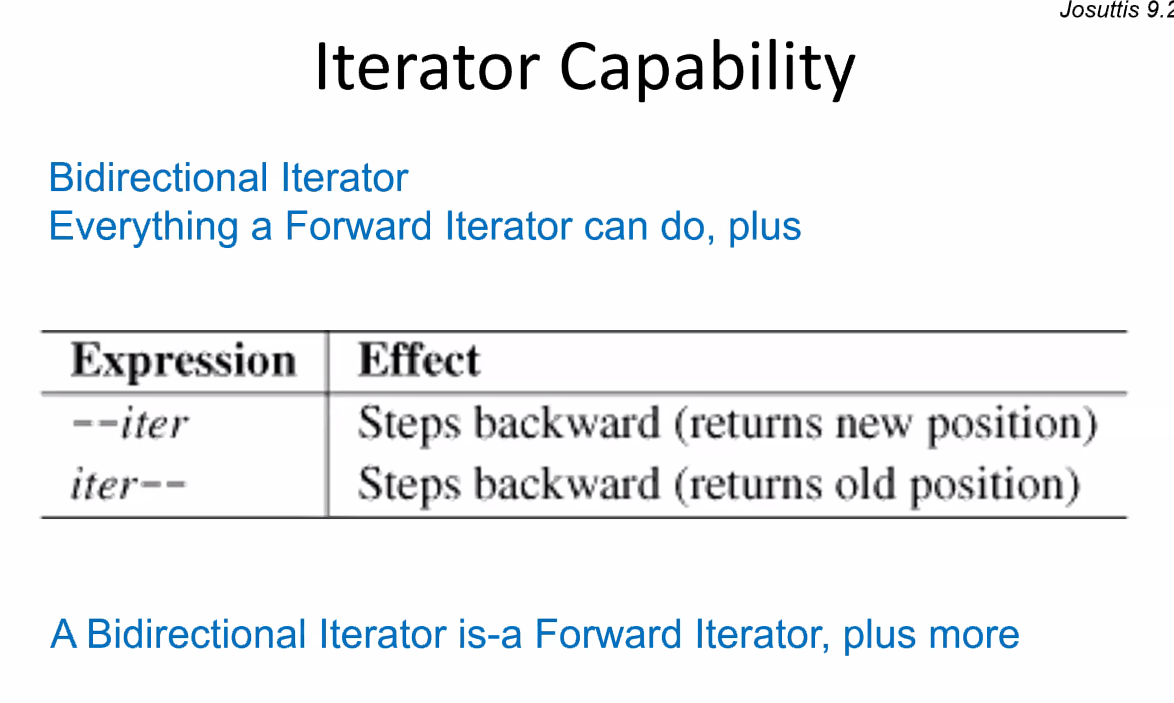
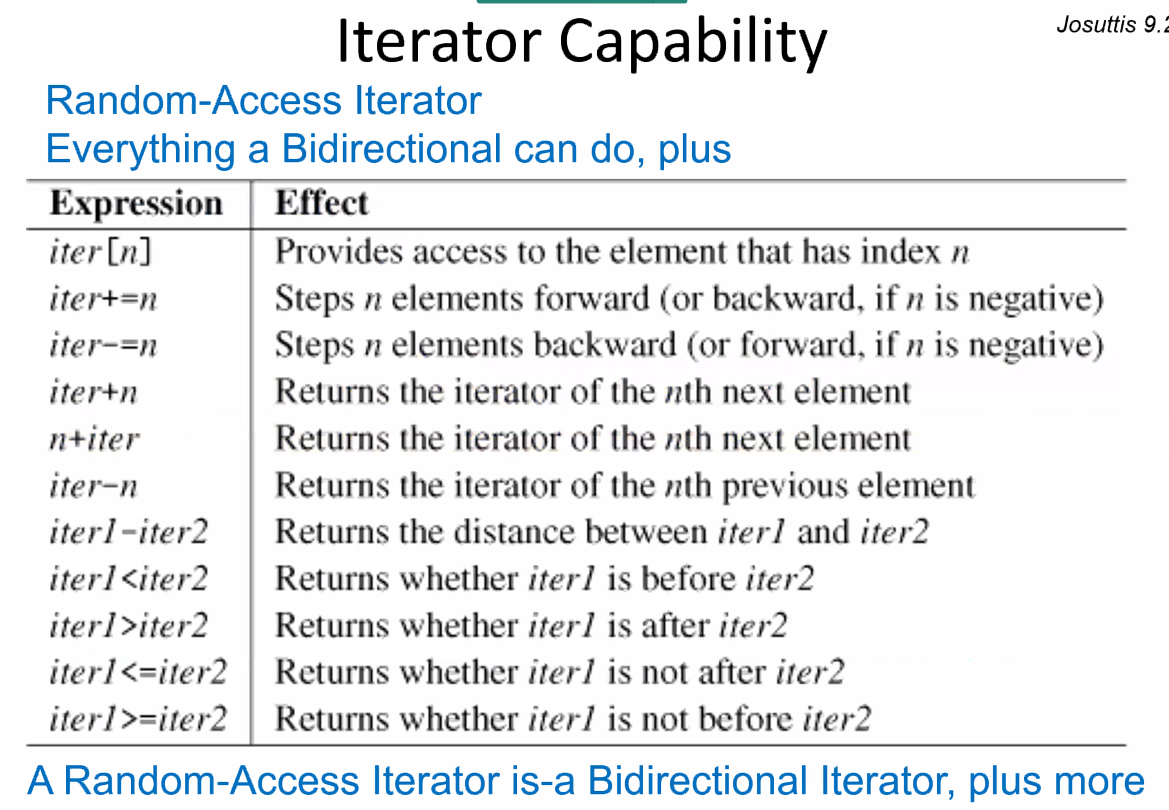
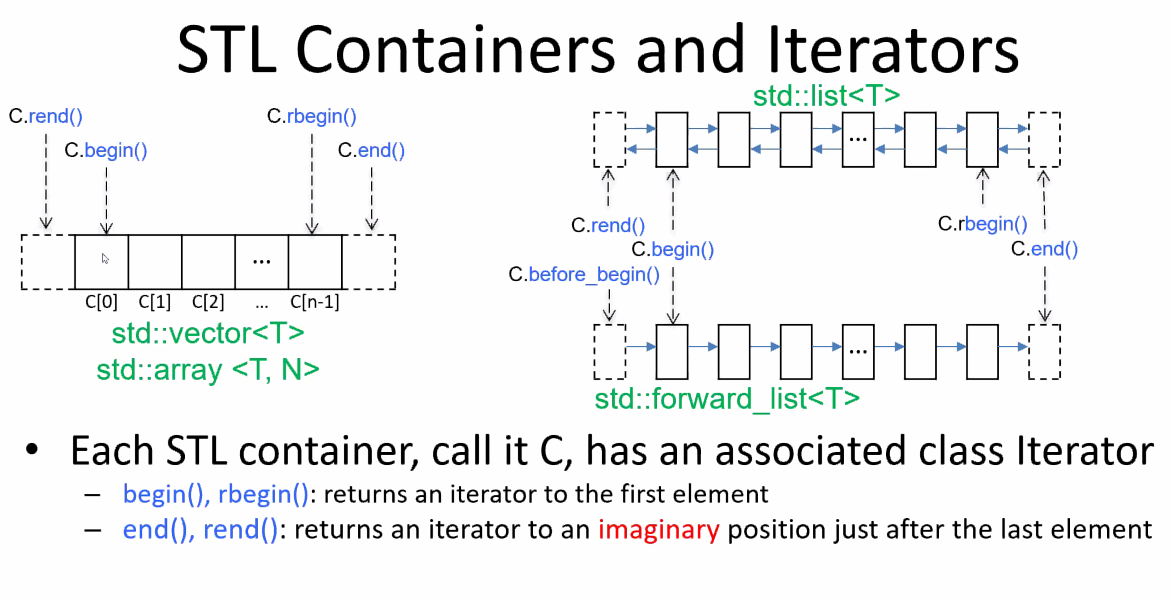
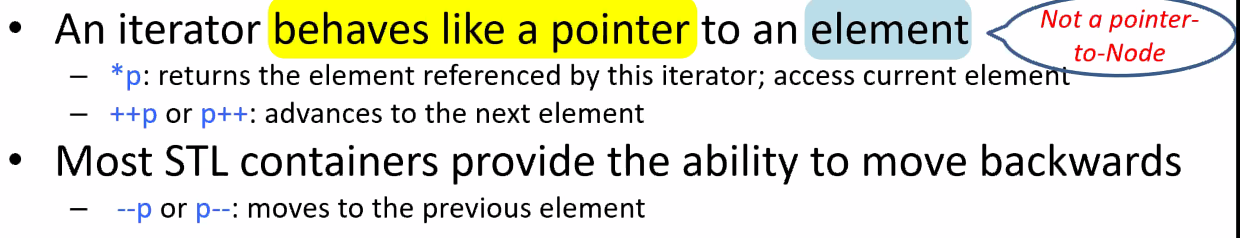
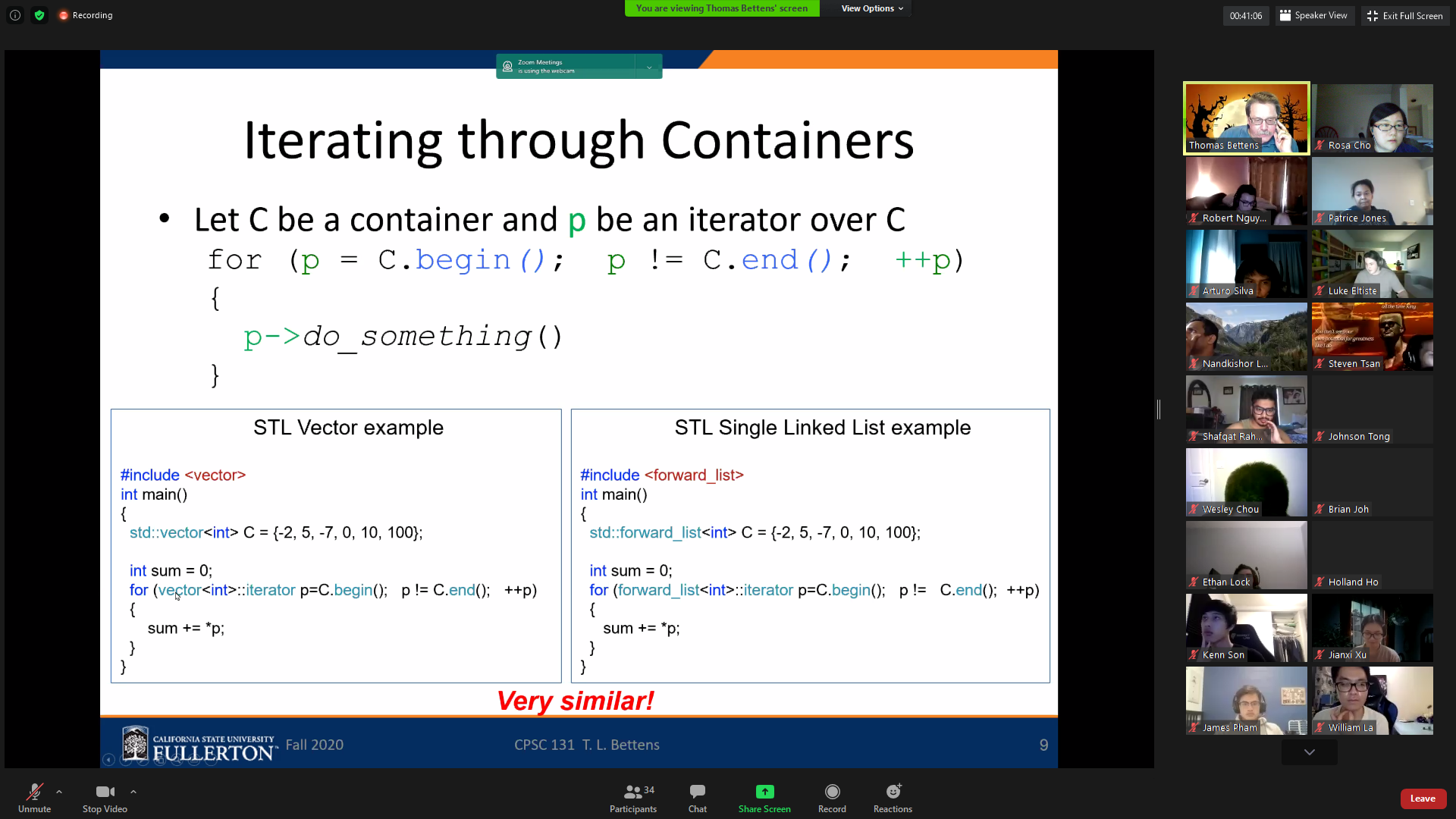
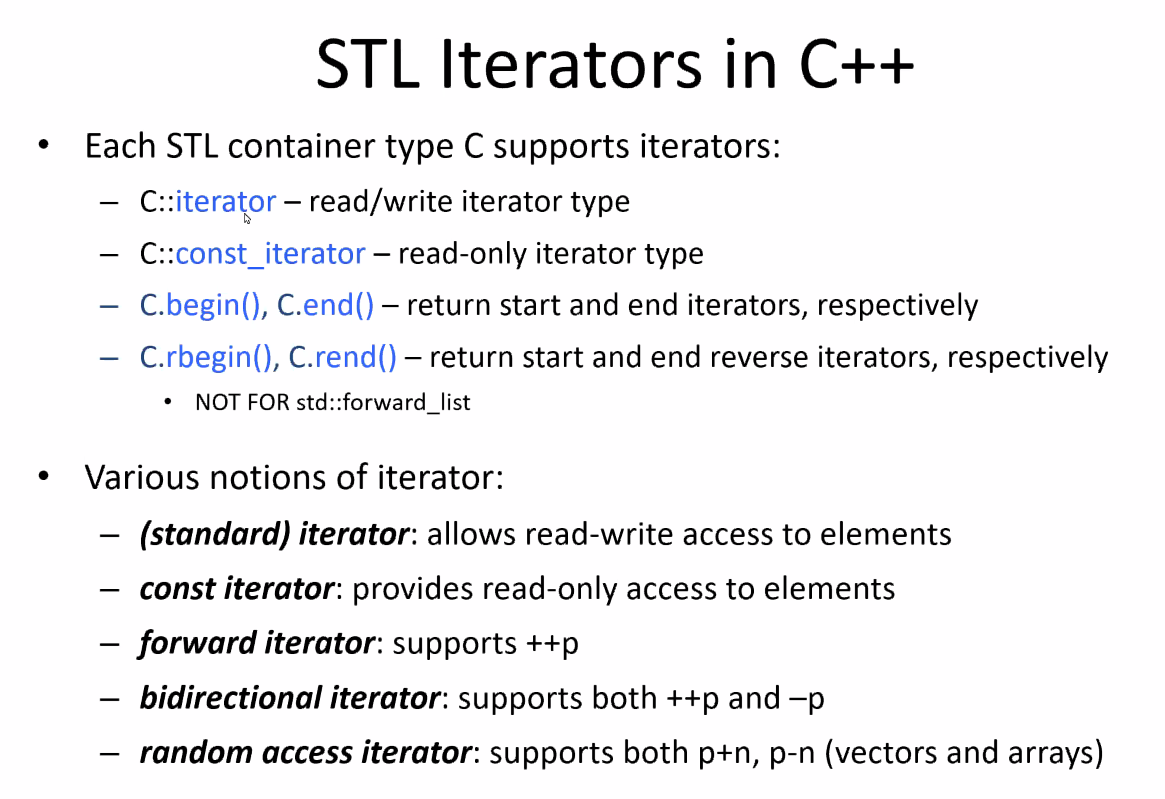
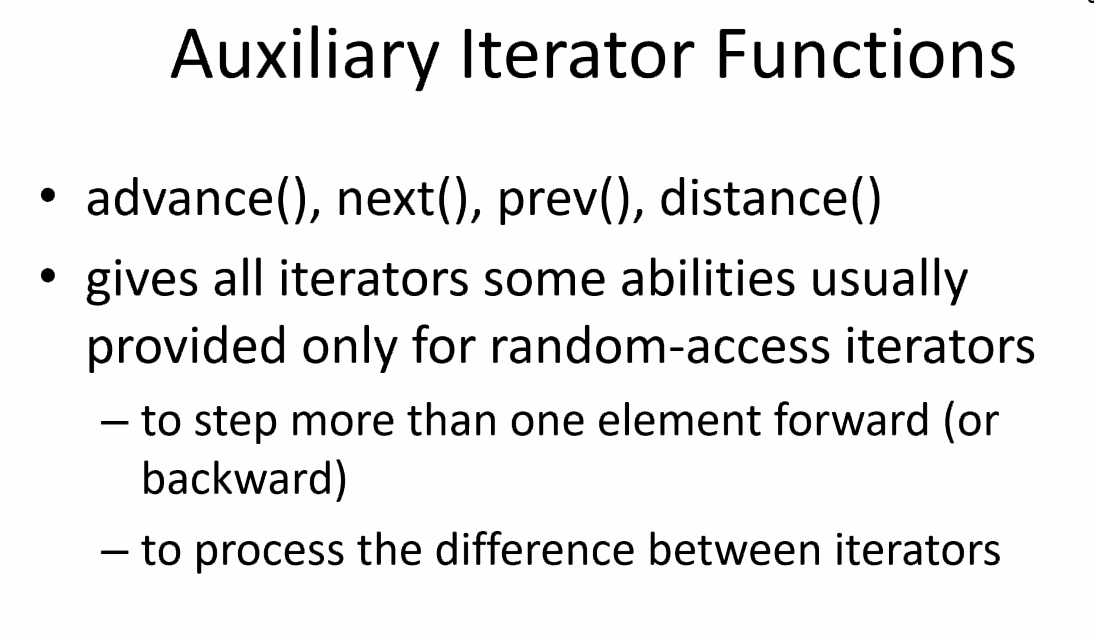
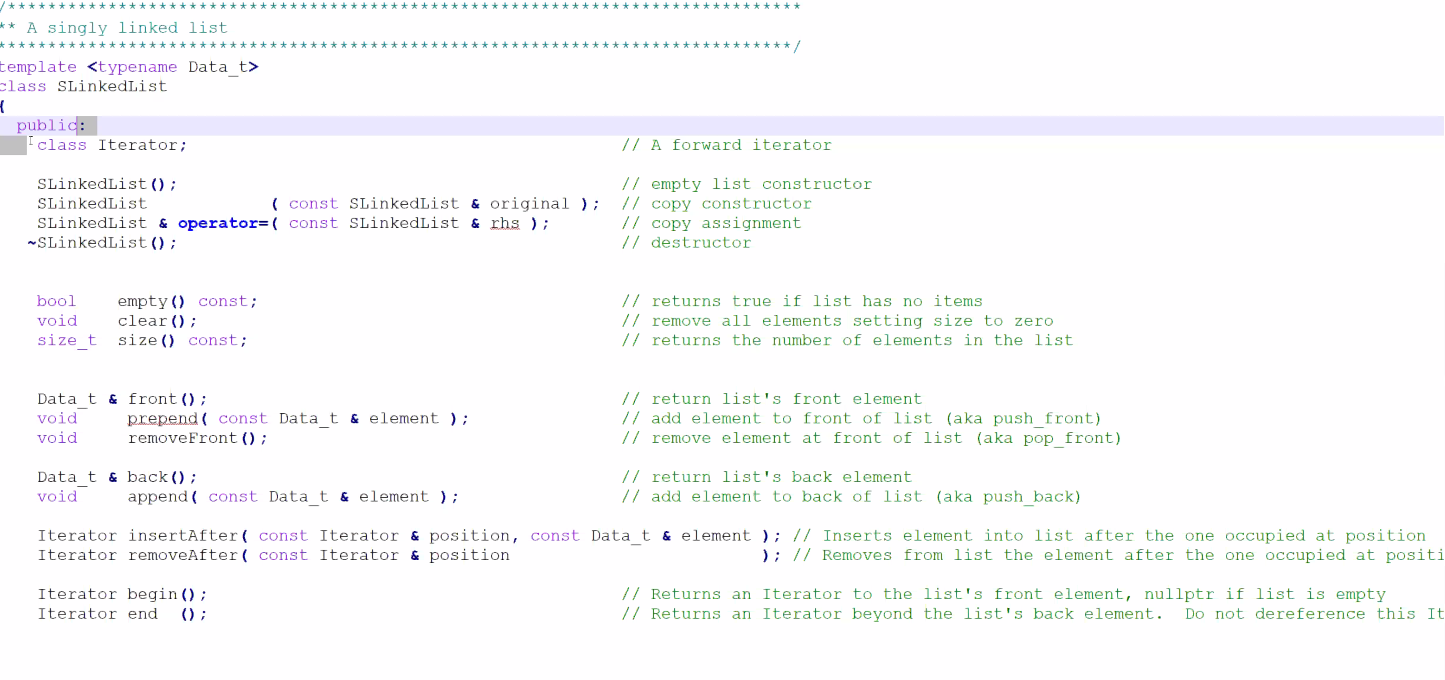
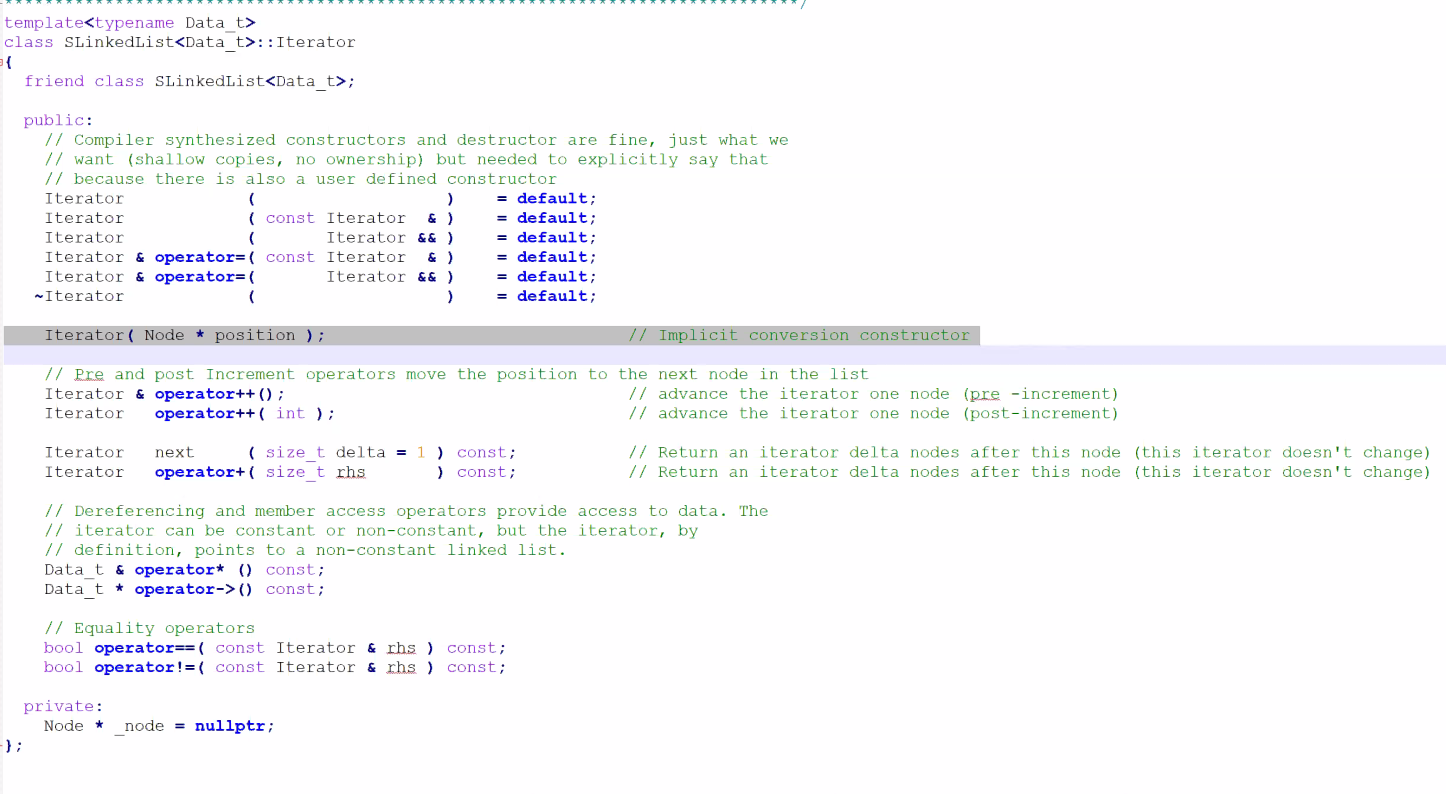
Lecture 10

9/28/2020

CPCSC 131

1. P2: BOOK DATABASE THE PROJECT
   1. We’ve been dealing with books for a while. Now, let’s go shopping!
   2. This time, we’re shuffling books from a “broken cart” and then put them into a working cart. However, to do that, we need to use a spare cart.
   3. In the end, we want all the books into the working cart.
   4. Essentially, using the Tower of Hanoi algorithm.
   5. A recursive algorithm to carefully move books from the broken cart to working cart is:  
      
      1. Implement find() recursively. It takes an iSBN, searches number of entries and segments of main
   6. Whate we given:
      1. Book interface
      2. Start with BookDatabase.cpp, hpp, and main
      3. Also test functions
2. Lecture: Iterators
   1. Containers and Iterators
      1. Container: abstract DS that stores collection of elements
      2. Iterator: abstracts the process of looping thorugh the collection of element
         1. Let C be a container and p be and iterator over C  
            for (p = C.begin(); p!=C.end(); ++p)  
            {  
            p->do\_something();  
            }
         2. Pre-increment > post increment
            1. It’s cheaper memory-wise when it comes to complicated situations
   2. Iterator Categories
      * 1. They are objects
           1. Can iterate over elements of a sequence via a common interface

What makes them special is that, if we can create an object such that incrementing meant advancing the pointers so many bytes, we won’t have to worry about the limitations linked lists (per say) have

* + - * 1. Adapted from ordinary pointers
      1. Anything that behaves like an iterator is an iterator
      2. Iterators have more abilities
  1. Iterator Categories  
     
     1. A generalized concept of operators
     2. Come in different flavors to match the flavors of data structures we have
  2. Iterator Capability
     1. 
  3. The BIdrectional Iterator
     1. 
     2. They can do anything a forward can but also go backwards
  4. Random Access Iterators
     1. 
     2. Both the forward and bidirectional did not have the < or > or any other relational operators
  5. STL Containers and Iterators
     1. 
     2. c.begin(), rbegin points to the first element. However rbegin goes in reverse.
     3. 
     4. Never pointer to node
     5. When you deref an interator, give access to the element within the node NOT the node
  6. Iterating through Containers  
     
  7. STL Iterators in C++
     1. 
     2. Each iterator is unique to the continaer it is linked to
     3. Difference between iterator and const\_interator
        1. It’s like a difference between a ptr and const\_ptr
        2. You can read and write in iterator but not the const
           1. Cannot edit what ht iterator points to
  8. Auxiliary Iterator Functions
     1. 
     2. Good place to start on Wednesday
  9. Coding Example: Singly Linked List Iterator
     1. 
     2. Haven’t defined the class but here’s the signature
     3. Introduces the name
     4. 
     5. IT defines the increment operators, next, derefs, equality
     6. You usually aren’t expected to code/define iterators in class but in real life…
     7. Pre vs Post. Pt.2
        1. Pre: change value before using it, Post: change value after using it
        2. Create temporary object, increment it, return the temporary object
     8. Iterator::next()
        1. 
     9. All this code is from the Sequential Container Implementation example files
     10. You can’t cmpare itearots but what theyr’e pointing to
  10. Operator ++  
      